Spectrales Clustering und Diffusionsabbildungen

von

Daniel Alexander Schmidt

Bachelorarbeit in Mathematik
vorgelegt dem Fachbereich Physik, Mathematik und Informatik (FB 08)
der Johannes Gutenberg-Universität Mainz

am 12. Juli 2022

1. Gutachter: Prof. Dr. Lisa Hartung
2. Gutachter: Prof. Dr. Achim Klenke

Zusammenfassung

Beim Clustering geht es darum, eine Menge von Datenpunkten ohne a priori Wissen in Gruppen einzuteilen. In dieser Arbeit beschäftigen wir uns mit Spectralem Clustering, ein Clusterverfahren, bei dem wir unsere Daten als Knoten eines Graphen auffassen und diesen Graphen versuchen, passend zu clustern. Wir werden so vorgehen, dass wir nach einer kurzen Einleitung in Kapitel 1 zunächst in Kapitel 2 grundlegend das Clusterproblem zusammen mit dem bekanntesten Clusteralgorithmus, dem k-means Algorithmus vorstellen, um danach elementare Graphentheorie gemeinsam mit verschiedenen Methoden, Daten in einen Graphen umzuwandeln, einzuführen. Wir werden uns dabei an [2] und [19] orientieren. Wie der Name vermuten lässt, bedient sich Spectrales Clustering an den Eigenvektoren und Eigenwerten bestimmter Repräsentationsmatrizen dieser Graphen, den sogenannten Laplace-Matrizen. Diese Matrizen und deren wichtigsten Eigenschaften, zusammen mit einem bekannten Spectralen Clusteralgorithmen aus [17], werden wir in Kapitel 3 vorstellen. In den darauffolgenden zwei Kapiteln werden wir zwei Interpretationen und Erklärungen dieses Algorithmus liefern. Kapitel 4 beschäftigt sich mit der approximierten Minimierung von Graphenschnitten, wohingegen Kapitel 5 den Zusammenhang zur Dimensionsreduktion mittels Diffusionsabbildungen herstellt. Schließlich werden wir in Kapitel 6 die Methoden zur Erstellung eines Graphen aus Kapitel 2 anhand von ausgewählten Beispielen diskutieren und diesbezüglich einige Faustregeln festlegen. Abschließend werden wir die Ergebnisse der Arbeit in Kapitel 7 zusammenfassen.

Inhaltsverzeichnis

1. Einleitung

„Information is the oil of the 21st century, and analytics is the combustion engine.“- Peter Sondergaard (2011)[18]. Dieses Zitat macht deutlicht, dass mit jährlich steigenden produziertem Datenvolumen weltweit [22] auch die Nachfrage steigt, aus diesen Daten Wissen zu generieren. Clustering ist dabei eine zentrale Aufgabe des sogenannten „unsupervised learning“, einem Teilgebiet des „machine learning“, bei dem es darum geht, Muster in Daten ohne a priori Wissen zu erkennen [3, S. 5 f.]. Clusterverfahren versuchen, Datenpunkte so zu gruppieren (in Cluster einzuteilen), dass Datenpunkte innerhalb einer Gruppe möglichst ähnlich und Datenpunkte verschiedener Gruppen möglichst unähnlich zueinander sind.

Als Spectrales Clustering bezeichnet man nun eine Gruppe von Verfahren, die alle ähnlich aufgebaut sind und grundlegend aus folgenden Schritten bestehen [10, S. 4]. Zunächst werden die Daten (wenn nicht schon passend gegeben) in einen Graphen umgewandelt. Dieser Graph soll die Ähnlichkeiten zwischen verschiedenen Datenpunkten modellieren, weshalb er auch Ähnlichkeitsgraph genannt wird. Wir stellen in dieser Arbeit verschiedene Methoden vor, aus einer Datenmenge einen Ähnlichkeitsgraphen zu erstellen, die sich an [19] orientieren. Danach berechnet man für den Graphen eine Laplace-Matrix, eine spezielle Form der Matrixrepräsentation eines Graphen. Verschiedene Spectrale Clusteralgorithmen unterscheiden sich dabei insofern voneinander, dass sie verschiedene Laplace-Matrizen berechnen. Als nächstes werden für die gewählte Laplace-Matrix die Eigenwerte und Eigenvektoren berechnet und jedem Graphenknoten eine niedrig dimensionale Repräsentation durch die Eigenvektoren zugeordnet. Schlussendlich erfolgt das Clustering durch Anwendung eines einfachen Clusteralgorithmus, wie beispielsweise k-means, auf die neue Repräsentation der Knoten.

Die Idee des Spectralen Clusterings entstand in den 1970er Jahren und geht zurück auf Donath und Hoffman (1973), die in [7] als erstes vorschlugen, Graphen anhand

der Eigenvektoren von Adjazenzmatrizen zu clustern. Im gleichen Jahr erforschte Fiedler in [8], dass eine Zweiteilung eines Graphen eng mit dem zweiten Eigenvektor der nicht normalisierten Laplace-Matrix verbunden ist. In der „machine learning community“ wurde die Spectrale Clusteranalyse anfang der 2000er besonders durch die Arbeit von Shi und Malik im Jahr 2000 [17] und Ng, Jordan und Weiss im Jahr 2002 [14] populär. Wir wollen in dieser Arbeit eine Version des Algorithmus aus [17] vorstellen. Von Luxburg lieferte zudem 2007 in [19] einen guten Überblick über die Spectrale Clusteranalyse mit mathematischen Hintergründen und praktischen Details.

Die Vorgehensweise von Spectralem Clustering hat den Vorteil, dass sie einfach zu implementieren ist, zusätzlich anders als viele herkömmliche Clusterverfahren auch nicht konvexe Strukturen in Daten erkennt und dabei nur klassische Methoden der Linearen Algebra verwendet.
Anwendung findet Spectrales Clustering beispielsweise in der Bildsegmentierung [17] oder in der Analyse sozialer Netzwerke [16].
Auf den ersten Blick kann es abwegig erscheinen, warum die Vorgehensweise von Spectralen Clusteralgorithmen zu einem guten Clustering führen soll. Ziel dieser Arbeit wird es deshalb sein, die mathematischen Grundlagen des Spectralen Clusteralgorithmus von [17] anhand von zwei Standpunkten zu erläutern und dabei die nicht triviale Wahl des Ähnlichkeitsgraphen mit entsprechenden Parametern zu diskutieren. Wir werden sehen, wie Graphenschnitte und die Dimensionsreduktion durch Diffusionsabbildungen eine tragende Rolle für den vorgestellten Algorithmus spielen und uns dabei größtenteils an [2] und [19] orientieren.

2. Das Clusterproblem und Ähnlichkeitsgraphen

Wie bereits angekündigt, ist es das Ziel von Clusteralgorithmen, eine Menge von Daten in sich ähnliche Gruppen zu teilen. Dabei benötigt man kein Vorwissen über die Daten. Für Anwendungsfälle denke man beispielsweise an eine Segmentierung der Kunden eines großen Unternehmens.

Im Folgenden werden wir zunächst den Fall betrachten, dass die Daten aus dem Euklidischen Raum stammen und einen der wohl bekanntesten Clusteralgorithmen für diesen Fall vorstellen, den k-means oder auch Lloyds Algorithmus [2, S. 52]. Dabei steht k für die Anzahl der zu findenden Cluster.

2.1. k-means

Gegeben seien $x_1, x_2, \ldots, x_n \in \mathbb{R}^p$. Ziel ist es nun, die Punkte in Gruppen $S_1, \ldots, S_k$ mit jeweiligen Zentren $\mu_1, \ldots, \mu_k \in \mathbb{R}^p$ zu unterteilen, sodass diese folgenden Ausdruck lösen:

$$\min_{\substack{S_1,\ldots,S_k \\ \mu_1,\ldots,\mu_k}} \sum_{l=1}^{k} \sum_{x_i \in S_l} \|x_i - \mu_l\|_2^2 \tag{2.1}$$

Dabei sollen die Gruppen oder auch Cluster paarweise disjunkt sein und die gesamte Datenmenge enthalten, also $S_1 \cup \cdots \cup S_k = \{x_1, \ldots, x_n\}$ und $S_i \cap S_j = \emptyset$ für $i \neq j$ erfüllen.
Die optimalen Zentren eines Clusters S_i sind dabei gegeben durch:

$$\mu_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j. \tag{2.2}$$

Allerdings ist die Lösung von (2.1) im Allgemeinen ein NP-schweres Problem [2]. Deshalb versuchen k-means Algorithmen, eine Lösung zu approximieren. Der am häufigsten verwendete k-means Algorithmus ist der sogenannte Lloyds Algorithmus, der auch oft als „der k-means Algorithmus“ bezeichnet wird [2, S. 52]. Dieser iterative Algorithmus funktioniert dabei folgendermaßen:

Algorithmus 2.1 (k-means [2, S. 52], [21])**.** *Eingabe:* $x_1, \ldots, x_n \in \mathbb{R}^p$ *und Anzahl der zu findenden Cluster* k*.*

1. *Wähle* k *zufällige Startzentren* $\mu_1, \ldots, \mu_k$*.*
2. *Weise jedem Datenpunkt dem Cluster zu, dessen Zentrum ihm am nächsten ist, also setze*
   $S_l := \{x_j : \|x_j - \mu_l\|_2 \leq \|x_j - \mu_i\|_2 \ \forall i = 1, \ldots, k\}, \ f\ddot{u}r\, l = 1, \ldots, k.$
3. *Aktualisiere Zentren* $\mu_1, \ldots, \mu_k$ *durch:*
   $\mu_l := \frac{1}{|S_l|} \sum_{x_j \in S_l} x_j, \ f\ddot{u}r\, l = 1, \ldots, k.$
4. *Wiederhole nun Schritt 2 und 3 solange, bis sich die Zuordnungen nicht mehr ändern.*

Obwohl dieser der wohl bekannteste Clusteralgorithmus ist, hat er dennoch einige Schwächen [2, S. 53], [21]. Zum einen hängt die gefundene Lösung stark von den anfangs gewählten Startpunkten ab. Zum anderen lässt sich k-means nur auf Daten anwenden, die sich im euklidischen Raum befinden.
Des Weiteren sind die durch den Algorithmus gefundenen Cluster aufgrund der Minimierung der Abstände zu den Clusterzentren immer konvex. Für Datenmengen wie beispielsweise in Abbildung 1 ist er deshalb ungeeignet.

Die Spectrale Clusteranalyse dagegen hat den Vorteil, dass sie auch nicht konvexe Strukturen in Daten erkennt [19, S. 24]. Beispielsweise kann sie die beiden Kreise in Abbildung 1 gut voneinander unterscheiden, indem die Daten zunächst mittels geeigneter Methoden in einen sogenannten Ähnlichkeitsgraphen umgewandelt werden (s. Abschnitt 2.3). Ein weiterer Vorteil des Spectralen Clusterings ist es, dass

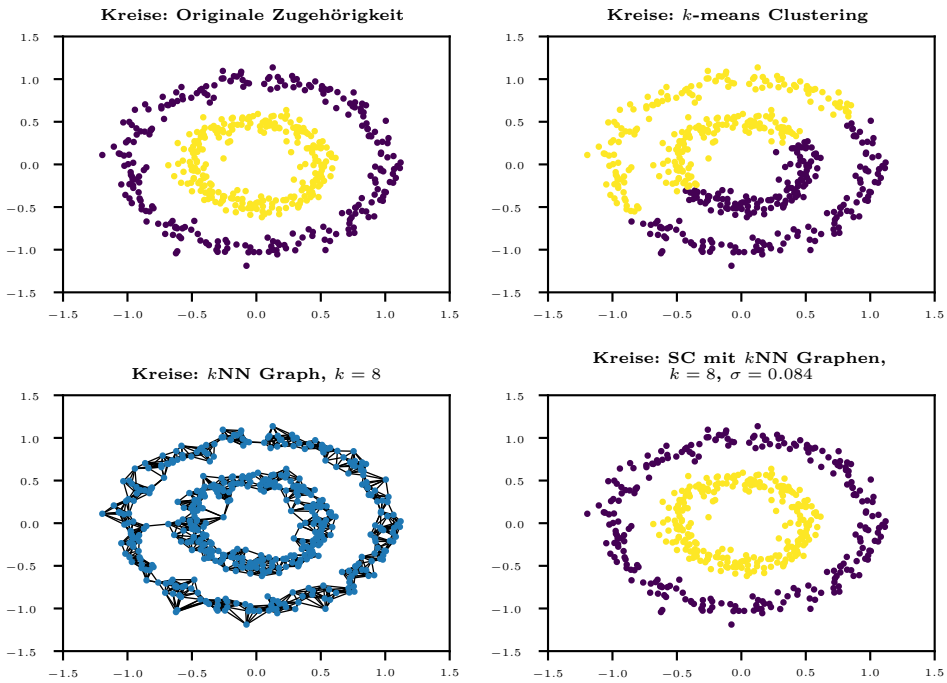


Abb. 1.: Vergleich von k-means und Spectralem Clustering bei nicht konvexen Daten: Die Daten bestehen aus 500 Punkten aufgeteilt auf zwei Kreise. Für jeden dieser Punkte wurde zusätzlich in beiden Dimensionen eine normalverteilte Zufallsvariable mit Mittelwert $\mu = 0$ und Varianz $\sigma^2 = 0.07^2$ aufaddiert.

die Daten beliebige Objekte sein können, für die nur eine paarweise Ähnlichkeit und kein Abstand im $\mathbb{R}^p$ gegeben sein muss.

Im Folgenden werden wir grundlegende Begriffe der Graphentheorie einführen, die wir im weiteren Verlauf der Arbeit verwenden werden.

2.2. Graphentheorie

Die anschließenden Definitionen orientieren sich grundlegend an [2, S. 50 f.], [19, S. 2], [6, Seite 1178 ff.].

Definition 2.2. *(1) Ein ungerichteter Graph $G = (V, E)$ ist ein Paar bestehend aus einer Menge $V = \{v_1, \ldots, v_n\}$ von Knoten (engl. vertices) und einer Menge $E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V \text{ und } v_i \neq v_j\}$ von Kanten (engl. edges) zwischen je zwei Knoten.*

(2) Ein Graph heißt zusammenhängend, wenn es für alle Paare von Knoten einen Pfad gibt, der diese verbindet.

(3) Für eine Teilmenge $U \subseteq V$ nennt man $G[U] := (U, \{\{v_i, v_j\} \in E : v_i, v_j \in U\})$ den von U induzierten Teilgraphen von G. Ein solcher zusammenhängender Teilgraph heißt Zusammenhangskomponente von G, wenn kein $W \subseteq V$ existiert mit $U \subsetneq W$ und $G[W]$ zusammenhängend.

Wir werden in dieser Arbeit oft die Kurzschreibweise $i \in U$ statt $v_i \in U$ für Teilmengen $U \subseteq V$ verwenden und bezeichnen mit $\mathbb{1}_U := \begin{pmatrix} f_1 & f_2 & \cdots & f_n \end{pmatrix}^\intercal \in \mathbb{R}^n$, mit $f_i = 1$, falls $v_i \in U$ und 0 sonst, den Indikatorvektor von $U \subseteq V$.

Ein gängiger Ansatz Graphen darzustellen, ist durch sogenannte Adjazenzmatrizen: Gegeben sei ein Graph $G = (V, E)$ mit $|V| = n$, dann definieren wir seine Adjazenzmatrix $A \in \mathbb{R}^{nxn}$ als symmetrische Matrix mit

$$A_{ij} = \begin{cases} 1 & \text{für } \{v_i, v_j\} \in E \\ 0 & \text{sonst} \quad . \end{cases}$$

Im Verlauf dieser Arbeit werden wir oft von gewichteten Graphen sprechen. Hier haben zwei Kanten ein entsprechendes Gewicht $w_{ij} = w_{ji} \geq 0$. Der Fall $w_{ij} = 0$ bedeutet hier ebenfalls, dass v_i und v_j nicht verbunden sind. Analog zur Adjazenzmatrix A definieren wir die gewichtete Adjazenzmatrix oder auch Gewichtsmatrix $W = (w_{ij})_{i,j=1,\ldots n}$.

2.3. Ähnlichkeitsgraphen

Wie bereits in Abschnitt 2.1 erwähnt, ist es beim Spectralen Clustering notwendig, eine gegebene Menge von Daten $x_1, \ldots, x_n$ mit paarweisen Ähnlichkeiten s_{ij} (oder Distanzen d_{ij}) zunächst in einen Graphen umzuwandeln. Dazu stellen wir in diesem Abschnitt verschiedene Möglichkeiten vor und werden diese anhand von

ausgewählten Beispielen in Kapitel 6 diskutieren. Wir werden uns dabei an [19, S. 2f., S. 17 ff.] orientieren. Ziel bei der Umwandlung unserer Daten in einen Graphen ist die Modellierung der lokalen Nachbarschaftsbeziehungen zwischen den Punkten. Dazu wählen wir zunächst als Knotenmenge $V = \{v_1, \ldots, v_n\}$ unseres Graphen die Punkte $x_1, \ldots, x_n$ selbst. Durch unsere Kantenmenge wollen wir nun die Ähnlichkeiten zwischen den Knoten repräsentieren. Dazu berechnen wir, wenn für unsere Daten paarweise Distanzen gegeben sind, meist zunächst Ähnlichkeitswerte mittels einer Ähnlichkeitsfunktion. Ein Beispiel für eine solche Funktion ist die Gaußsche Ähnlichkeitsfunktion

$$s(x_i, x_j) := \exp\left(\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}\right) = \exp\left(\frac{-{d_{ij}}^2}{2\sigma^2}\right), \tag{2.3}$$

mit der wir für Daten aus dem $\mathbb{R}^p$ Ähnlichkeitswerte erhalten. Eine hohe Distanz führt dann zu einer niedrigen Ähnlichkeit und eine niedrige Distanz zu einer hohen Ähnlichkeit. Im Fall der Gaußschen Ähnlichkeitsfunktion gilt, dass $0 < s(x_i, x_j) \leq 1$. Der Parameter σ steuert, ab wann die Ähnlichkeitswerte für größere Distanzen fallen. Je kleiner σ ist, desto schneller gehen die Ähnlichkeitswerte für große Abstände gegen null. Die Kantenmenge kann nun auf verschiedene Arten festlegt werden:

Der ϵ-Nachbarschaftsgraph (ϵ-Graph): Hier wählen wir zunächst ein $\epsilon > 0$, um dann alle Knoten miteinander zu verbinden, deren paarweise Distanz $d_{ij} < \epsilon$ oder deren paarweise Ähnlichkeite $s_{ij} > \epsilon$ ist. Da sich nun alle d_{ij} oder s_{ij} in der Größenordnung ϵ oder kleiner befinden, rät von Luxburg in [19] dazu, den ϵ-Nachbarschaftsgraph ungewichtet zu betrachten. Dieser Empfelung werden wir in dieser Arbeit folgen.

k-Nächste-Nachbarn-Graph (kNN-Graph): Beim KNN Graphen verbinden wir Knoten v_i mit Knoten v_j, wenn v_j zu den k nächsten Nachbarn von v_i gehört, also zu den k Knoten, die zu v_i die höchsten Ähnlichkeiten haben. Da dies im Allgemeinen nicht zu einem ungerichteten Graphen führt, verbinden wir v_i und v_j, falls v_j zu den k nächsten Nachbarn von v_i oder wenn v_i zu den k nächsten Nachbarn von v_j gehört. Jeder Knoten hat dann mindestens k Kanten, die wir durch die paarweisen Ähnlichkeiten gewichten.

Der voll verbundene Graph: Hier verbinden wir alle Knotenpunkte miteinander und gewichten alle Kanten durch die paarweisen Ähnlichkeiten s_{ij}.

3. Laplace-Matrizen und Spectrales Clustering

Beim Spectralen Clustering geht es darum, die Knoten des nach einen der Methoden aus Abschnitt 2.3 erzeugten Ähnlichkeitsgraphen in eine vorher festgelegte Anzahl k an Clustern zu partitionieren. Eine Partition der Knotenmenge V ist dabei analog wie im $\mathbb{R}^n$ definiert als Zerlegung von V in disjunkte Teilmengen $A_1, \ldots, A_k$, für die $A_1 \cup \cdots \cup A_k = V$ und $A_i \cap A_j = \emptyset$ für $i \neq j$ gilt.
Beim Spectralen Clustering greifen wir auf die Eigenwerte und Eigenvektoren der sogenannten Laplace-Matrizen zurück, um eine sinnvolle Partition der Knotenmenge eines Graphen zu finden. Es sei angemerkt, dass es in der Literatur keine einheitliche Konvention gibt, welche Matrix „die Laplace-Matrix“ ist. Wir folgen an dieser Stelle den Notationen aus [19].

3.1. Nicht normalisierte Laplace-Matrix

Im Folgenden betrachten wir immer einen ungerichteten Graphen $G = (V, E, W)$ mit Gewichts- oder Adjazenzmatrix $W = (w_{ij})_{i,j=1,\ldots,n}$ mit $w_{ij} \geq 0$. Wenn wir in diesem Abschnitt von den Eigenwerten sprechen, gehen wir immer davon aus, dass diese unter Berücksichtigung der algebraischen Multiplizitäten aufsteigend sortiert sind. Mit „den ersten k Eigenvektoren“, meinen wir jene, welche zu den k-kleinsten Eigenwerten gehören.

Definition 3.1 (Gradmatrix und nicht normalisierte Laplace-Matrix)**.**

(1) Der Grad eines Knoten v_i ist definiert als

$$d_i := \sum_{j=1}^{n} w_{ij}. \tag{3.1}$$

Als Gradmatrix

$$D := \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix} \tag{3.2}$$

bezeichnen wir die Diagonalmatrix mit dem jeweiligen Graden der Knoten auf der Hauptdiagonalen.

(2) Die nicht normalisierte Laplace-Matrix ist definiert durch

$$L := D - W. \tag{3.3}$$

Nun werden wir kurz einige nützliche Eigenschaften von L darstellen.

Proposition 3.2 (Eigenschaften von L [19, Prop. 1])**.** *Für L aus* (3.3) *gelten folgende Eigenschaften.*

(1) Für $f \in \mathbb{R}^n$ gilt:

$$f^\intercal L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \tag{3.4}$$

(2) L ist symmetrisch und positiv semidefinit.

(3) Für die Eigenwerte $\{\lambda_i\}_{i=1,\dots,n}$ von L gilt:

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

und ein Eigenvektor zum Eigenwert $\lambda_1 = 0$ ist der konstante Vektor.

Beweis. (1): Zunächst gilt für $f \in \mathbb{R}^n$:

$$f^\intercal L f = \sum_{i,j=1}^{n} L_{ij} f_i f_j \overset{(3.3)}{=} \sum_{i,j=1}^{n} D_{ij} f_i f_j - W_{ij} f_i f_j.$$

Nutzen erhalten wir mit der Definition von D aus (3.2):

$$
\begin{aligned}
\sum_{i,j=1}^{n} D_{ij} f_i f_j - W_{ij} f_i f_j &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} w_{ij} f_i f_j \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} d_i f_i^2 - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n} d_i f_i^2 \right) \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} d_i f_i^2 - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{j=1}^{n} d_j f_j^2 \right) \\
&\overset{(3.1)}{=} \frac{1}{2} \left(\sum_{i=1}^{n} f_i^2 \sum_{j=1}^{n} w_{ij} - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{j=1}^{n} f_j^2 \sum_{i=1}^{n} w_{ji} \right) \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_i^2 w_{ij} - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n} \sum_{j=1}^{n} f_j^2 w_{ij} \right) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2,
\end{aligned}
$$

wobei wir im vorletzten Schritt die Symmetrie von W, also $w_{ij} = w_{ji}$ und im letzten Schritt die 2. binomische Formel benutzt haben.
(2): Die Symmetrie von L gilt nach Definition, da D und W symmetrisch sind. Außerdem gilt nach (3.4), dass f positiv semidefinit ist, da $w_{ij} \geq 0$ und somit $f^\intercal L f \geq 0$ für belieges $f \in \mathbb{R}^n$.
(3): Wir erhalten dann, dass L aufgrund seiner Symmetrie n und positiven Semidefinitheit n nicht negative Eigenwerte besitzt. Sei $\mathbf{1} := \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^\intercal \in \mathbb{R}^n$ nun der konstante Eins-Vektor . Dann gilt wieder mit der Definition von d_i für beliebiges $1 \leq i \leq n$:

$$
(L\mathbf{1})_i = \sum_{j=1}^{n} L_{ij} = \sum_{j=1}^{n} -w_{ij} \; + d_i = 0.
$$

Also ist $\mathbf{1}$ ein Eigenvektor zum Eigenwert $\lambda_1 = 0$ und die Behauptung folgt. □

3.2. Normalisierte Laplace-Matrizen

Nachdem wir im letzten Abschnitt die nicht normalisierte Laplace-Matrix betrachtet haben, befassen wir uns nun mit den normalisierten Laplace-Matrizen, dessen

Eigenwerte und Eigenvektoren für den Spectralen Clusteralgorithmus, den wir vorstellen werden, von großer Bedeutung sind. Wir gehen dabei davon aus, dass jeder Knoten v_i mindestens eine Kante besitzt, sodass $d_i > 0$ und sowohl D^{-1} als auch $D^{-\frac{1}{2}}$ existieren. Die normalisierten Laplace-Matrizen sind folgendermaßen definiert:

Definition 3.3 (L_{sym} *und* L_{rw})**.** *Wir bezeichnen mit*

$$L_{sym} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_d - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \quad \textit{und} \tag{3.5}$$

$$L_{rw} := D^{-1} L = I_d - D^{-1} W \tag{3.6}$$

die normalisierten Laplace-Matrizen eines ungerichteten Graphen $G = (V, E, W)$ *mit Gewichtsmatrix* W *und Gradmatrix* D*.*

Die Namen orientieren sich dabei an den Eigenschaften der Matrizen. L_{sym} ist nach Definition eine symmetrische Matrix und L_{rw} ist eng verwandt mit der Übergangsmatrix einer zufälligen Irrfahrt (engl. random walk) auf einem Graphen. Auf diesen Zusammenhang kommen wir in Kapitel 5 noch einmal zurück.

Auch für diese Matrizen wollen wir zunächst gewisse Eigenschaften festhalten.

Proposition 3.4 (Spektrale Eigenschaften von L_{sym} und L_{rw} [19, Prop. 3])**.** *Für* L_{sym} *und* L_{rw} *aus* (3.5) *und* (3.6) *gelten folgende Eigenschaften.*

(1) Für $f \in \mathbb{R}^n$ *gilt:*

$$f^\intercal L_{sym} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 . \tag{3.7}$$

(2) λ *ist ein Eigenwert von* L_{rw} *mit zugehörigem Eigenvektor* u *genau dann, wenn* λ *ein Eigenwert von* L_{sym} *mit zugehörigem Eigenvektor* $D^{\frac{1}{2}} u$ *ist.*

(3) λ *ist ein Eigenwert von* L_{rw} *mit zugehörigem Eigenvektor* u *genau dann, wenn* λ *und* u *das allgemeine Eigenwertproblem* $Lu = \lambda Du$ *lösen.*

(4) 0 ist ein Eigenwert von L_{rw} *mit zugehörigem Eigenvektor* $\mathbf{1}$ *(der konstante Eins-Vektor) und ein Eigenwert von* L_{sym} *mit Eigenvektor* $D^{\frac{1}{2}} \mathbf{1}$*.*

(5) L_{sym} *und* L_{rw} *sind positiv semidefinit und besitzen* n *nicht negative reelle Eigenwerte* $0 = \lambda_1 \leq \cdots \leq \lambda_n$.

Beweis. (1): Der Beweis geht analog zu (1) aus Proposition 3.2.
(2): Sei λ ein Eigenwert von L_{sym} mit Eigenwert w. Dann gilt:

$$L_{sym}w = \lambda w \overset{\cdot D^{-\frac{1}{2}}}{\Longrightarrow} D^{-\frac{1}{2}}L_{sym}w = D^{-1}LD^{-\frac{1}{2}}w = \lambda D^{-\frac{1}{2}}w$$

$$\Longrightarrow L_{rw}D^{-\frac{1}{2}}w = \lambda D^{-\frac{1}{2}}w \Longrightarrow L_{rw}u = \lambda u,$$

wobei wir $u = D^{-\frac{1}{2}}w$ im letzten Schritt substituiert haben. Analog funktioniert die Rückrichtung mit Multiplikation von $D^{\frac{1}{2}}$ von links.
(3): Betrachten wir die Eigenwertgleichung für L_{rw}, dann gilt:

$$L_{rw}u = \lambda u \overset{\cdot D}{\Longrightarrow} DD^{-1}Lu = \lambda Du \Longrightarrow Lu = \lambda Du.$$

Die Rückrichtung folgt wieder analog durch Multiplikation mit D^{-1} von links.
(4): Es gilt:

$$L_{rw}\mathbf{1} = D^{-1}L\mathbf{1} \overset{\text{Prop.3.2}}{=} D^{-1}\mathbf{0} = \mathbf{0}.$$

Die Aussage für L_{sym} folgt direkt mit Punkt 2.
(5): Für L_{sym} erhalten wir die positiv Semidefinitheit durch (1), da $f^\intercal L_{sym}f \geq 0$. Insbesondere wissen wir dann, dass alle Eigenwerte von L_{sym}, die aufgrund der Symmetrie reell sind, nicht negativ sind. Nach Punkt (4) wissen wir außerdem, dass $\lambda_1 = 0$. Damit erhalten wir die Aussage für L_{sym}. Für L_{rw} folgt die Aussage dann nach Punkt (2) . □

Proposition 3.5 (Zusammenhang Spektren und Anzahl der Zusammenhangskomponenten [19, Prop. 4])**.** *Sei* G *ein ungerichteter Graph. Dann entspricht die algebraische Vielfachheit* k *des Eigenwerts* $\lambda_1 = 0$ *von sowohl* L_{sym} *als auch* L_{rw} *der Anzahl der Zusammenhangskomponenten* $A_1, \ldots, A_k$ *des Graphen. Für* L_{rw} *wird der Eigenraum des Eigenwerts* 0 *von Indikatorvektoren* $\mathbb{1}_{A_i}$ *der Zusammenhangskomponenten aufgespannt. Für* L_{sym} *wird der Eigenraum des Eigenwerts 0 entsprechend von* $D^{\frac{1}{2}}\mathbb{1}_{A_i}$ *aufgespannt.*

Beweis. Ein Beweis einer analogen Aussage für L findet sich in [19, Prop. 2]. □

3.3. Spectrales Clustering - Ein Algorithmus

In diesem Abschnitt möchten wir einen der bekanntesten Spectralen Clusteralgorithmen vorstellen, der auf der Berechnung der Eigenvektoren der normalisierten Laplace-Matrizen L_{rw} beruht. Man nennt ihn deshalb auch Normalisiertes Spectrales Clustering.
An dieser Stelle sei erwähnt, dass es auch einen unnormalisierten Spectralen Clusteralgorithmus gibt, der die Eigenvektoren der unnormalisierten Laplace-Matrix L benutzt. In [19] wird dieser ausführlich erklärt.

Algorithmus 3.6 (Normalisiertes Spectrales Clustering nach [17], [19])**.** *Eingabe: Ähnlichkeitsmatrix $S \in \mathbb{R}^{nxn}$ und Anzahl k der zu findenden Cluster.*

1. *Konstruiere zunächst einen Ähnlichkeitsgraphen durch einer der in Abschnitte 2.3 beschriebenen Methoden. Sei W die Gewichtsmatrix dieses Graphen.*
2. *Berechne die unnormalisierte Laplace-Matrix L.*
3. *Berechne die ersten k Eigenvektoren $u_1, \ldots, u_k$ des allgemeinen Eigenwertproblems $Lu = \lambda Du$.*
4. *Sei $U \in \mathbb{R}^{nxk}$ die Matrix mit den Eigenvektoren $u_1, \ldots, u_k$ als Spalten. Definiere $y_i \in \mathbb{R}^k$ als die i-te Zeile von U, für $i = 1, \ldots, n$.*
5. *Cluster die Punkte $\{y_1, \ldots, y_n\}$ in $\mathbb{R}^k$ mittels des k-means Algorithmus in Cluster $C_1, \ldots, C_k$.*
6. *Erhalte Knotenpartition $A_1, \ldots, A_k$, wobei $A_i = \{v_j | y_j \in C_i\}$.*

Wir sehen, dass Algorithmus 3.6 die Eigenvektoren des allgemeinen Eigenwertproblems $Lu = \lambda Du$ nutzt, was nach Proposition 3.4 genau den Eigenvektoren der normalisierten Laplace-Matrix L_{rw} entspricht.

Bemerkung 3.7. Es gibt Versionen dieses Algorithmus, bei denen wir statt der Eigenvektoren $u_1, \ldots, u_k$ der normalisierten Laplace-Matrix L_{rw} nur $u_2, \ldots, u_k$ verwenden, um aus diesen Vektoren die Matrix U zu bilden. Grund dafür ist,

dass der erste Eigenvektor von L_{rw} ein Vielfaches des konstanten Vektors $\mathbf{1}$ ist (s. Prop. 3.4). Dieser Vektor trägt deshalb nicht dazu bei, die Knoten voneinander zu unterscheiden.

Wesentlicher Bestandteil dieses Algorithmus ist es, die Darstellung der Punkte x_i beziehungsweise der Knoten v_i zu Punkten y_i aus dem $\mathbb{R}^k$ zu ändern. Auf den ersten Blick kann einem diese Vorgehensweise fragwürdig erscheinen und man kann sich durchaus die Frage stellen, warum die Anwendung von k-means auf Zeilen der Eigenwertsmatrix U von L_{rw} eine sinnvolle Art zum Clustern ist. Tatsächlich werden wir auf diese Frage in den nächsten Kapiteln sinnvolle Antworten finden, indem wir zwei Arten, den Algorithmus zu interpretieren, kennenlernen.

Zunächst wollen wir uns aber an einem einfachen Beispiel die Aussagekraft der Eigenvektoren der normalisierten Laplace-Matrix L_{rw} illustrieren.

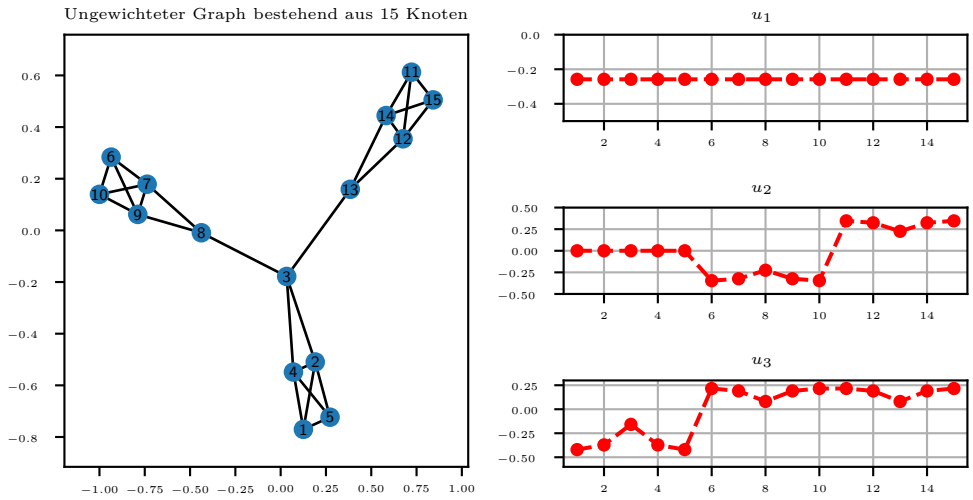


Abb. 2.: Dreifaches Haus des Nikolaus mit ersten drei Eigenvektoren von L_{rw}.

Betrachten wir dazu den ungewichteten Graphen aus Abbildung 2. Dieser besteht aus drei „Häusern des Nikolaus“, die über die jeweiligen Spitzen miteinander verbunden sind. Daneben sehen wir die ersten drei Eigenvektoren der normalisierten Laplace-Matrix L_{rw} des Graphen. Für einen Eigenvektor $u_i = \begin{pmatrix} u_{1,i} & \cdots & u_{15,i} \end{pmatrix}^\intercal$ haben wir jeweils $u_{j,i}$ gegen j geplotted. Für einen Knoten v_j steht nach Algorithmus

3.6 jeweils der j-te Eintrag der drei Eigenvektoren repräsentativ für diesen Knoten. Zunächst halten wir fest, dass der Graph zusammenhängend ist. Der erste Eigenvektor u_1 ist wie erwartet konstant und die Anwendung von k-means auf diesen leistet keinen Beitrag, die Knoten des Graphen zu unterscheiden. Der zweite und dritte Eigenvektor dagegen liefern wesentliche Informationen über den Graphen. Aus dem zweiten Eigenvektor u_2 wird erkenntlich, dass die Einträge für Knoten $v_6, \ldots, v_{10}$ negativ, für Knoten $v_{11}, \ldots, v_{15}$ gleichermaßen positiv und für Knoten $v_1, \ldots, v_5$ konstant null sind. Insbesondere können wir dadurch Knoten $v_6, \ldots, v_{10}$ von den restlichen unterscheiden. Für u_3 sehen wir, dass die Einträge für $v_1, \ldots, v_5$ kleiner null und für Knoten $v_6, \ldots, v_{15}$ größer null sind. Hier können wir also explizit die ersten fünf Knoten von den restlichen abgrenzen. Wenn wir nun die gewonnenen Erkenntnisse durch die Eigenvektoren u_2 und u_3 zusammenführen, erhalten wir die intuitive Aufteilung des Graphen in Cluster $A_1 = \{v_1, \ldots, v_5\}$, $A_2 = \{v_6, \ldots, v_{10}\}$, $A_3 = \{v_{11}, \ldots, v_{15}\}$, also genau in die drei „Häusern des Nikolaus“. Wir sehen also, dass wir mit der Analyse der ersten drei Eigenvektoren der normalisierten Laplace-Matrix L_{rw} des Graphen wesentliche Informationen über des Graphen erhalten, mit denen wir die Knotenmenge des Graphen einfach partitionieren können.

4. Graphenschnitte

Das allgemeine Ziel beim Clustering ist die Einteilung von Datenpunkten in sich ähnliche Gruppen. Im Fall, dass unsere Daten Knoten eines Ähnlichkeitsgraphen $G = (V, E, W)$ sind, können wir das Clustering-Problem folgendermaßen formulieren [19, S.8]: Wir wollen eine Partition $A_1, \ldots, A_k$ der Knotenmenge V finden, sodass Kanten zwischen verschiedenen Gruppen ein geringes Kantengewicht haben (sich also möglichst unähnlich sind) und Kanten innerhalb einer Gruppe möglichst große Gewichte haben (sich also möglichst ähnlich sind). In diesem Abschnitt werden wir Graphenschnitte als Möglichkeiten zur Messung der Güte einer Partition einführen und anhand der approximierten Minimierung einer dieser Schnitte, die folgen werden, dem $Ncut$, den Spectralen Clusteralgorithmus aus Kapitel 3 erklären. Wir orientieren uns dabei an dem Vorgehen aus [2, S. 54 ff.] und [19, S. 8 ff.].

Gegeben sei wie im letzten Kapitel immer ein gegebenfalls gewichteter ungerichteter Graph $G = (V, E, W)$. Die einfachste Art, eine Partition zu messen, ist der Graphenschnitt.

Definition 4.1 (Schnitt)**.** *Sei $A_1, \ldots, A_k$ eine Partition der Knotenmenge V eines Graphen $G = (V, E, W)$. Dann ist der Schnitt (engl. cut) der Partition definiert als*

$$cut(A_1, \ldots, A_k) := \frac{1}{2} \sum_{l=1}^{k} \sum_{\substack{i \in A_l \\ j \in A_l^c}} w_{ij}. \tag{4.1}$$

Für den Fall $k = 2$ entspricht die Definition genau:

$$cut(A, A^c) = \frac{1}{2} \left(\sum_{\substack{i \in A \\ j \in A^c}} w_{ij} + \sum_{\substack{i \in A^c \\ j \in A}} w_{ij} \right) = \sum_{\substack{i \in A \\ j \in A^c}} w_{ij}. \tag{4.2}$$

Man kann sich den Schnitt so vorstellen, dass man alle Kanten, die zwischen

Knoten verschiedener Gruppen verlaufen, durchschneidet und dabei ihre Gewichte aufaddiert. Je geringer dann die resultierende Summe ist, desto unähnlicher sind sich die Gruppen.

Der Einfachheit halber wollen wir uns zunächst auf den Fall $k = 2$, also auf eine Zweiteilung des Graphen beschränken.

4.1. Der Fall $k = 2$

Im Fall von zwei Clustern ist der Ansatz, $cut(A, A^c)$ zu minimieren, ein relativ leichtes effizient zu lösendes Problem, das jedoch meist nicht zu zufriedenstellenden Ergebnissen führt ([2, S. 55], [19, S. 8]). Betrachten wir dazu Abbildung 3, ein Beispiel eines Graphen bestehend aus Knotenmenge $\{v_0, \ldots, v_{10}\}$. Diesen Graphen wollen wir nun in zwei Cluster partitionieren.

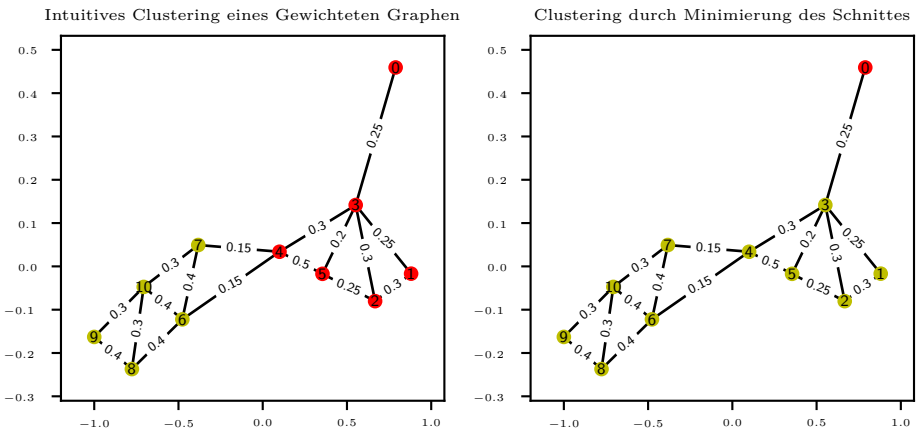


Abb. 3.: Minimaler Graphenschnitt eines gewichteten Graphen.

Auf der linken Seite sehen wir eine Zweiteilung des Graphen in zwei ähnlich große Gruppen, wie vermutlich viele den Graphen „per Auge “ einteilen würden. Nennen wir diese Partition $A_1 = \{v_0, \ldots v_5\}, A_1^c = \{v_6, \ldots, v_{10}\}$. Der Schnitt dieser Aufteilung entspricht dann

$$cut(A_1, A_1^c) = w_{47} + w_{46} = 0.15 + 0.15 = 0.3.$$

Auf der rechten Seite sehen wir die Aufteilung des Graphen, die den Schnitt minimiert. Dabei gilt $A_2 = \{v_1, \ldots, v_{10}\}$ und $A_2^c = \{v_0\}$ mit

$$cut(A_2, A_2^c) = w_{03} = 0.25 < 0.3 = cut(A_1, A_1^c).$$

Der minimale Schnitt wird hier also durch die Abspaltung eines einzelnen Knoten erreicht. Dies ist nicht das Ergebnis, welches wir uns beim Clustern, bei dem wir einigermaßen ähnliche Gruppen identifizieren wollen, erhoffen. Deshalb stellen wir nun einen alternativen Schnitt vor, der die Ausgeglichenheit der Gruppen miteinbezieht.

Definition 4.2 (Normalisierter Schnitt)**.** *Sei A, A^c eine Partition eines Graphen $G = (V, E, W)$ und $vol(A) := \sum_{i \in A} d_i$, dann heißt*

$$Ncut(A, A^c) := \frac{cut(A, A^c)}{vol(A)} + \frac{cut(A, A^c)}{vol(A^c)} \tag{4.3}$$

der normalisierte Schnitt (engl. normalized cut).

Der normalisierte Schnitt misst nach Definition die Ausgeglichenheit einer Partition durch die Summe der Kantengewichte zugehörig zu den Knoten der Gruppen.

Der Vollständigkeit halber sei erwähnt, dass es auch andere Möglichkeiten gibt, die Ausgeglichenheit eines Graphenschnitts miteinzubeziehen, beispielsweise durch die Anzahl der Knoten in den einzelnen Clustern. Eine Möglichkeit dies umzusetzen, bietet der sogenannte „*RatioSchnitt* “(engl. *Ratiocut*), dessen Minimierung in [19] zum unnormaliserten Spectralen Clustering führt.

In dem, was nun folgt, wollen wir uns auf das Finden einer Partition, die den $Ncut$ minimert, konzentrieren. Zunächst halten wir fest, dass sich $Ncut$ als quadratische Form mit der nicht normalisierten Laplace-Matrix L schreiben lässt:

Proposition 4.3 ([2, Prop. 4.6 und 4.7], [19, S. 10 f.])**.** *Sei A, A^c eine Partition der Knotenmenge V von $G = (V, E, W)$. Dann gelten folgende Aussagen.*

(1) Es gilt:

$$Ncut(A, A^c) = f^\intercal L f, \tag{4.4}$$

wobei $f \in \mathbb{R}^n$ ein Indikatorvektor der Partition A, A^c ist gegeben durch

$$f_i := \begin{cases} \left(\frac{vol(A^c)}{vol(A)vol(G)}\right)^{\frac{1}{2}} & \text{für } v_i \in A \\ -\left(\frac{vol(A)}{vol(A^c)vol(G)}\right)^{\frac{1}{2}} & \text{für } v_i \in A^c\,. \end{cases} \tag{4.5}$$

(2) Für einen Indikatorvektor $f \in \{a, b\}^n$ mit $a, b \in \mathbb{R}$ einer Partition A, A^c gilt genau dann $\mathbf{1}^\intercal D f = 0$ und $f^\intercal D f = 1$, wenn f wie in (4.5) (oder mit umgekehrten Vorzeichen) definiert ist.

Beweis. (1): Sei also $f \in \mathbb{R}^n$ wie in (4.5) definiert. Dann gilt zunächst mit der Definition von f und der Symmetrie von W, also $w_{ij} = w_{ji}$:

$$\begin{aligned} f^\intercal L f &\overset{(3.4)}{=} \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \\ &= \frac{1}{2}\sum_{i\in A,\, j\in A^c} w_{ij}(f_i - f_j)^2 + \frac{1}{2}\sum_{i\in A^c,\, j\in A} w_{ij}(f_i - f_j)^2 \\ &= \sum_{i\in A,\, j\in A^c} w_{ij}\left[\left(\frac{vol(A^c)}{vol(A)vol(G)}\right)^{\frac{1}{2}} + \left(\frac{vol(A)}{vol(A^c)vol(G)}\right)^{\frac{1}{2}}\right]^2 \\ &= \sum_{i\in A,\, j\in A^c} w_{ij}\,\frac{1}{vol(G)}\left[\left(\frac{vol(A^c)}{vol(A)}\right)^{\frac{1}{2}} + \left(\frac{vol(A)}{vol(A^c)}\right)^{\frac{1}{2}}\right]^2 =: (*). \end{aligned}$$

Mit der ersten Binomischen Formel folgt dann:

$$\begin{aligned} (*) &= \sum_{i\in A,\, j\in A^c} w_{ij}\,\frac{1}{vol(G)}\left[\frac{vol(A^c)}{vol(A)} + \frac{vol(A)}{vol(A^c)} + 2\right] \\ &= \sum_{i\in A,\, j\in A^c} w_{ij}\,\frac{1}{vol(G)}\left[\frac{vol(A^c)}{vol(A)} + \frac{vol(A)}{vol(A^c)} + \frac{vol(A)}{vol(A)} + \frac{vol(A^c)}{vol(A^c)}\right] \\ &= \sum_{i\in A,\, j\in A^c} w_{ij}\,\frac{1}{vol(G)}\left[\frac{vol(G)}{vol(A)} + \frac{vol(G)}{vol(A^c)}\right] \\ &= \sum_{i\in A,\, j\in A^c} w_{ij}\left[\frac{1}{vol(A)} + \frac{1}{vol(A^c)}\right] \\ &\overset{(4.2)}{=} cut(A, A^c)\left[\frac{1}{vol(A)} + \frac{1}{vol(A^c)}\right] \\ &\overset{(4.3)}{=} Ncut(A, A^c), \end{aligned}$$

wobei wir genutzt haben, dass $vol(A) + vol(A^c) = vol(G)$.

(2): Für die Hinrichtung sei f durch $f_i = \begin{cases} a & \text{für } v_i \in A \\ b & \text{für } v_i \in A^c \end{cases}$ gegeben.

Dann folgt:

$$\begin{aligned} \mathbf{1}^\intercal Df &= \sum_{i=1}^{n} d_i f_i = \sum_{i \in A} d_i f_i + \sum_{i \in A^c} d_i f_i \\ &= a \sum_{i \in A} d_i + b \sum_{i \in A^c} d_i = a\, vol(A) + b\, vol(A^c) \end{aligned}$$

und außerdem

$$\begin{aligned} f^\intercal Df &= \sum_{i \in A} d_i f_i^2 + \sum_{i \in A^c} d_i f_i^2 \\ &= a^2\, vol(A) + b^2\, vol(A^c). \end{aligned}$$

Um $\mathbf{1}^\intercal Df = 0$ und $f^\intercal Df = 1$ zu erhalten, müssen wir also das Gleichungssystem

$$\begin{aligned} a\, vol(A) + b\, vol(A^c) &= 0 \\ a^2\, vol(A) + b^2\, vol(A^c) &= 1 \end{aligned}$$

lösen.

Aus der ersten Gleichung erhalten wir unter Berücksichtigung, dass $A \neq \emptyset$ und somit $vol(A) \neq 0$

$$a = -b\, \frac{vol(A^c)}{vol(A)}.$$

Dies in die zweiten Gleichung eingesetzt, ergibt

$$\begin{aligned} & \left(-b\, \tfrac{vol(A^c)}{vol(A)}\right)^2 vol(A) + b^2\, vol(A^c) && = 1. \\ \iff \quad & b^2 \left[vol(A^c)^2 + vol(A) vol(A^c)\right] && = vol(A) \\ \iff \quad & b^2 && = \frac{vol(A)}{vol(A^c)^2 + vol(A) vol(A^c)} \\ \iff \quad & b^2 && = \frac{vol(A)}{vol(A^c)\left(vol(A) + vol(A^c)\right)} \\ \iff \quad & b^2 && = \frac{vol(A)}{vol(A^c) vol(G)}. \end{aligned}$$

Wir erhalten also $b_{1/2} = \mp \left(\frac{vol(A)}{vol(A^c)vol(G)}\right)^{\frac{1}{2}}$ und $a_{1/2} = \pm \left(\frac{vol(A^c)}{vol(A)vol(G)}\right)^{\frac{1}{2}}$, wodurch der Vektor unter Berücksichtigung der Vorzeichen genau f aus (4.5) entspricht.

Für die Rückrichtung sei f also durch (4.5) gegeben, dann gilt:

$$\begin{aligned}
\mathbf{1}^\intercal Df &= \sum_{i=1}^{n} d_i f_i = \sum_{i\in A} d_i f_i + \sum_{i\in A^c} d_i f_i \\
&= \sum_{i\in A} d_i \left(\frac{vol(A^c)}{vol(A)vol(G)}\right)^{\frac{1}{2}} - \sum_{i\in A^c} d_i \left(\frac{vol(A)}{vol(A^c)vol(G)}\right)^{\frac{1}{2}} \\
&= \left(\frac{vol(A^c)}{vol(A)vol(G)}\right)^{\frac{1}{2}} vol(A) - \left(\frac{vol(A)}{vol(A^c)vol(G)}\right)^{\frac{1}{2}} vol(A^c) \\
&= 0
\end{aligned}$$

und außerdem:

$$\begin{aligned}
f^\intercal Df &= \sum_{i=1}^{n} f_i d_i f_i \\
&= \sum_{i\in A} f_i^2 d_i + \sum_{j\in A^c} f_j^2 d_j \\
&= \left(\frac{vol(A^c)}{vol(A)vol(G)}\right) vol(A) + \left(\frac{vol(A)}{vol(A^c)vol(G)}\right) vol(A^c) \qquad = 1,
\end{aligned}$$

wobei wir im letzten Schritt wieder $vol(A) + vol(A^c) = vol(G)$ benutzt haben.

□

Mit vorheriger Proposition haben wir also gezeigt, dass wir eine Minimierung des $Ncut$ umschreiben können zu:

$$\min_{A,A^c} Ncut(A, A^c) = \min_{\substack{f\in\{a,b\}^n \text{ für gewisse } a,b \\ \mathbf{1}^\intercal Df=0 \\ f^\intercal Df=1}} f^\intercal Lf. \tag{4.6}$$

Shi und Malik zeigen in [17, S. 901 ff.], dass die Optimierung dieses Problems im Allgemeinen NP-schwer ist. Aus diesem Grund werden wir das Problem etwas vereinfachen. Ein Ansatz ist es, die Diskretheitsbedingung des Vektors f zu verwerfen und f stattdessen zu erlauben, beliebige Werte anzunehmen. Diese Relaxation

führt zu dem entspannten Optimierungsproblem

$$\min_{\substack{f\in\mathbb{R}^n \\ \mathbf{1}^\intercal Df=0 \\ f^\intercal Df=1}} f^\intercal L\, f. \tag{4.7}$$

Wenn wir nun wir $g := D^{\frac{1}{2}} f$ substituieren, lautet das Problem

$$\min_{\substack{g\in\mathbb{R}^n \\ \left(D^{\frac{1}{2}}\mathbf{1}\right)^\intercal g=0 \\ \|g\|_2=1}} g^\intercal D^{-\frac{1}{2}} L D^{-\frac{1}{2}} g \;=\; \min_{\substack{g\in\mathbb{R}^n \\ \left(D^{\frac{1}{2}}\mathbf{1}\right)^\intercal g=0 \\ \|g\|_2=1}} g^\intercal L_{sym}\, g. \tag{4.8}$$

Durch Anwendung des Satzes von Courant-Fischer (s. [13, 5.2.2.(4)]) und unter Berücksichtigung, dass der kleinste Eigenvektor von L_{sym} durch $D^{\frac{1}{2}}\mathbf{1}$ gegeben ist, wird klar, dass wir eine Lösung von (4.8) durch den zweiten Eigenvektor von L_{sym} erhalten. Wenn wir nun $f = D^{-\frac{1}{2}} g$ rücksubstituieren und Punkt 2 aus Proposition 3.4 anwenden, sehen wir, dass die Lösung von (4.7) durch den zweiten Eigenvektor von L_{rw} gegeben ist.
Da es das ursprüngliche Ziel war, durch f eine Partition mit möglichst geringen normalisiertem Schnitt zu finden, müssen wir den reelen Lösungsvektor f von (4.7) noch in einen diskreten Indikatorvektor verwandeln. Bandeira, Singer und Strohmer schlagen in [2, S. 59] als Möglichkeit der Diskretisierung vor, eine Schranke $\tau \in \mathbb{R}$ zu wählen und A durch $A = \{v_i \in V : f_i \leq \tau\}$ zu definieren. Im Fall $k = 2$ könne man beispielsweise alle n Möglichkeiten ausprobieren und jene Partition wählen, die den geringsten Normalisierten Schnitt liefere. Im diesem Fall ist diese Vorgehensweise durchaus ein sinnvoller Ansatz.
Im Allgemeinen Fall $k > 2$ benötigt es aber eine andere Heuristik [19, S.9 ff.]. Die meisten Spectralen Clusteralgorithmen betrachten stattdessen die Koordinaten f_i als Punkte in $\mathbb{R}$, clustern diese durch den k-means Algorithmus in zwei Gruppen C, C^c und übertragen die resultierende Cluster auf die zugrundeliegenden Daten durch

$$\begin{cases} v_i \in A & \text{falls } f_i \in C \\ v_i \in A^c & \text{falls } f_i \in C^c. \end{cases}$$

Das entspricht unter Berücksichtigung von Bemerkung 3.7 genau der Vorgehensweise des vorgestellten Algorithmus 3.6 für den Fall $k = 2$.

4.2. Der allgemeine Fall $k \geq 2$

In diesem Abschnitt beleuchten wir den Fall, dass wir einen Graphen in eine beliebige Anzahl $2 \leq k < n$ partitionieren wollen. Dazu betrachten wir wieder den normalisierten Schnitt.

Definition 4.4. *Sei $A_1, \ldots, A_k$ eine Partition des Graphen G=(V,E,W). Dann definieren wir den normalisierten Schnitt als*

$$Ncut(A_1, \ldots, A_k) := \sum_{i=1}^{k} \frac{cut(A_i, A_i^c)}{vol(A_i)}. \tag{4.9}$$

Diese Definition entspricht für $k = 2$ genau der Definition aus (4.3).

Ähnlich wie im Fall $k = 2$ definieren wir uns für jedes A_j einen Indikatorvektor $h_j = \begin{pmatrix} h_{1,j} & \cdots & h_{n,j} \end{pmatrix}^\intercal$ durch

$$h_{i,j} = \begin{cases} \frac{1}{\sqrt{vol(A_j)}} & \text{falls } v_i \in A_j \\ 0 & \text{sonst} \end{cases} \quad (i = 1, \ldots, n;\ j = 1, \ldots, k). \tag{4.10}$$

Setzen wir H als die Matrix, die diese k Indikatorvektoren als Spalten enthält, also

$$H = \begin{pmatrix} h_{1,1} & \cdots & h_{1,k} \\ \vdots & & \vdots \\ h_{n,1} & \cdots & h_{n,k}, \end{pmatrix}, \tag{4.11}$$

so stellen wir Folgendes fest:

Proposition 4.5. *Sei H wie in* (4.10) *und* (4.11) *definiert, dann gilt:*

(1) $H^\intercal D H = I_d$

(2) $Spur(H^\intercal L H) = Ncut(A_1, \ldots, A_k)$.

Beweis. (1): Für $i \neq j$ gilt:

$$h_j^\intercal D h_i = \sum_{k=1}^{n} h_{k,j} d_k h_{k,i} = 0,$$

da $A_1, \ldots, A_k$ eine Partition bilden, also $A_i \cap A_j = \emptyset$ für $i \neq j$ und somit entweder $h_{k,j} = 0$ oder $h_{k,i} = 0$.
Für $j = i$ dagegen gilt:

$$h_j^\intercal D h_i = \sum_{k=1}^{n} h_{k,i}^2 d_k = \sum_{k \in A_i} \frac{d_k}{vol(A_i)} = 1.$$

Also folgt, dass $H^\intercal D H = I_d$.
(2): Da für $v_k, v_j \in A_i : h_{k,i} - h_{j,i} = 0$, gilt:

$$\begin{aligned} h_i^\intercal L h_i &\overset{(3.4)}{=} \frac{1}{2} \sum_{k,j=1}^{n} w_{kj}(h_{k,i} - h_{j,i})^2 = \frac{1}{2} \left(\sum_{\substack{k \in A_i \\ j \in A_i^c}} \frac{w_{kj}}{vol(A_i)} + \sum_{\substack{k \in A_i^c \\ j \in A_i}} \frac{w_{kj}}{vol(A_i)} \right) \\ &= \sum_{\substack{k \in A_i^c \\ j \in A_i}} \frac{w_{kj}}{vol(A_i)} = \frac{cut(A_i, A_i^c)}{vol(A_i)}. \end{aligned}$$

Damit folgt

$$Ncut(A_1, \ldots, A_k) \overset{(4.9)}{=} \sum_{i=1}^{n} \frac{cut(A_i, A_i^c)}{vol(A_i)} = \sum_{i=1}^{n} h_i^\intercal L h_i = Spur(H^\intercal L H).$$

□

Wir haben also gezeigt, dass wir die Minimierung des normalisierten Schnitts wieder umschreiben können:

$$\min_{A_1,\ldots,A_k} Ncut(A_1, \ldots, A_k) = \min_{\substack{A_1,\ldots,A_k \\ H \text{ wie in (4.10) und (4.11)} \\ H^\intercal D H = I_d}} Spur(H^\intercal L H). \tag{4.12}$$

Dieses Problem ist genau wie Problem (4.6) NP-schwer [17, S. 901 ff.], weshalb wir wieder eine Relaxation des Problems betrachten, indem wir die Diskretheitsbedin-

gung von H aufheben. Dadurch erhalten wir:

$$\min_{\substack{H\in\mathbb{R}^{nxk}\\ H^\intercal DH=I_d}} Spur(H^\intercal LH). \tag{4.13}$$

Nun substituieren wir $T := D^{\frac{1}{2}}H$ und erhalten

$$\min_{\substack{T\in\mathbb{R}^{nxk}\\ T^\intercal T=I_d}} Spur(T^\intercal D^{-\frac{1}{2}}LD^{-\frac{1}{2}}T) = \min_{\substack{T\in\mathbb{R}^{nxk}\\ T^\intercal T=I_d}} Spur(T^\intercal L_{sym}T). \tag{4.14}$$

Dieses Problem ist ein Standard Spur-Minimierungsproblem, für das die Lösung nach [13, 5.2.2.(6)] durch die Matrix T gegeben ist, die die ersten k Eigenvektoren von L_{sym} als Spalten enthält. Analog zum Fall $k = 2$ sehen wir, dass durch Rücksubstitution und Anwendung von Proposition 3.4 folgt, dass H in (4.13) durch die ersten k Eigenvektoren von L_{rw} gegeben ist. H enthält also die Eigenvektoren, die wir in Algorithmus 3.6 benutzen.
Wäre H nun die Partitionsmatrix aus (4.10) und (4.11), so wäre genau der Eintrag H_{ij} der i-ten Zeile von H ungleich null, für den Knoten v_i in Cluster A_j läge. Wir könnten aus H also direkt eine Partition der Knotenmenge ablesen. Da nun H als Lösung von (4.13) eine reele Lösungsmatrix ist, müssen wir in einem Nachbereitungsschritt, analog wie im Fall $k = 2$, die Lösungsmatrix H rückdiskretisieren, um eine eindeutige Knotenpartition zu erhalten. Zur Rückdiskretisierung der reellen Lösungsmatrix H aus des Problems (4.13) wenden wir deshalb k-means auf die Zeilen $y_1, \ldots, y_k \in \mathbb{R}^k$ der Matrix H an und erhalten Cluster $C_1, \ldots, C_k$, die wir wie im Fall von zwei Clustern auf die zugrundeliegenden Knoten übertragen:

$$A_i = \{v_j \mid y_j \in C_i\}, \quad i = 1, \ldots, k.$$

Das beschriebene Vorgehen entspricht genau Algorithmus 3.6 aus Abschnitt 3.3.

5. Diffusionsabbildungen und Irrfahrtsansatz

Wie bereits in Abschnitt 3.2 angedeutet, ist die Matrix L_{rw} eng verwandt mit dem Begriff der Irrfahrt (engl. random walks) auf einem Graphen. Genau diese Verbindung wollen wir in diesen Kapitel herstellen und dabei den Zusammenhang von Spectralem Clustering zu Diffusionsabbildungen, einer Methode zur nichtlinearen Dimensionsreduktion, deutlich machen. Mit Diffusionsabbildungen können wir insbesondere auch Graphen, die im Allgemeinen keine räumliche Lage haben, in den Euklidischen Raum einbetten. Wir folgen dabei Kapitel 5 aus [2].

5.1. Graphenirrfahrt und Diffusionsabbildungen

Wir betrachten nun eine Irrfahrt mit zugehöriger Markovkette $(X_t)_{t\in\mathbb{N}}$ auf einem Graphen $G = (V, E, W)$ mit Gewichten $w_{ij} \geq 0$. Unser Zustandsraum ist V und wir definieren die Wahrscheinlichkeit von Knoten v_i zu Knoten v_j zu springen als relativen Anteil des Kantengewichts w_{ij} am Grad des Knoten v_i:

$$P(X_{t+1} = v_j \mid X_t = v_i) = \frac{w_{ij}}{d_i}.$$

Dann ist die Übergangsmatrix $M = (M_{ij})_{i,j=1,\dots,n}$ unserer Markovkette definiert durch

$$M_{ij} = \frac{w_{ij}}{d_i}.$$

Sei D wie in den vorherigen Kapiteln die Gradmatrix des Graphen, dann gilt also

$$M = D^{-1}W.$$

Wenn wir nun unsere Irrfahrt in einem Knoten v_i des Graphen starten (also $X_0 = v_i$), dann gilt (vgl. [4, Korollar 7.4]) für die Wahrscheinlichkeit, dass die Irrfahrt nach t Schritten auf Knoten v_j landet

$$P(X_t = v_j \mid X_0 = v_i) = (M^t)_{ij}. \tag{5.1}$$

Die Wahrscheinlichkeitsverteilung der Irrfahrt zur Zeit t mit Start in v_i ist also durch den Zeilenvektor

$$P(X_t \mid X_0 = v_i) = e_i^\intercal M^t = \begin{pmatrix} M_{i1}^t & \cdots & M_{in}^t \end{pmatrix} =: M^t[i,:] \tag{5.2}$$

gegeben.
Zur Einbettung eines Graphen in den euklidischen Raum kann man nun den Ansatz verfolgen, einem Knoten seiner Wahrscheinlichkeitsverteilung nach t Schritten zuzuordnen. Somit würde man Knoten, die nach t Schritten eine ähnliche Verteilung haben, nahe beieinander platzieren. Einen ähnlichen Ansatz werden wir in diesem Kapitel durch Diffusionsabbildungen nachgehen.
Bevor wir diesen Ansatz weiter verfolgen, beschäftigen wir uns zunächst mit den Eigenwerten und Eigenvektoren der Matrix M. Ähnlich zu den letzten Kapiteln ordnen wir in diesem Kapitel wieder die Eigenwerte mit zugehörigen Eigenvektoren von M, dieses mal allerdings absteigend, also $\lambda_1 \geq \cdots \geq \lambda_n$. Wir bertachten nun folgendes nützliches Resultat über die Eigenwerte und Eigenvektoren von M, das gleichzeitig die Verwandtschaft von M zu L_{rw} verdeutlicht.

Proposition 5.1 (Spektrale Eigenschaften von M [2, S. 66 f.], [19, S. 12])**.** *Sei $M = D^{-1}W$ die Übergangsmatrix der Irrfahrt auf einem Graphen $G = (V, E, W)$ mit normalisierter Laplace-Matrix L_{rw}, dann gelten folgende Aussagen.*

(1) λ ist ein Eigenwert mit zugehörigem Eigenvektor φ von L_{rw} genau dann, wenn $1 - \lambda$ ein Eigenwert mit Eigenvektor φ von M ist.

(2) Wir können M zerlegen in

$$M = \Phi \Lambda \Psi^\intercal, \tag{5.3}$$

wobei $\Phi = \begin{pmatrix} \varphi_1 & \cdots & \varphi_n \end{pmatrix}$, $\Psi = \begin{pmatrix} \psi_1 & \cdots & \psi_n \end{pmatrix}$ und Λ die Diagonalmatrix mit den Eigenwerten $\lambda_1, \ldots, \lambda_n$ von M in absteigender Reihenfolge auf der

Hauptdiagonalen ist. φ_k und ψ_k sind dabei jeweils die Rechts- und Linkseigenvektoren von M, konkret gilt also für $1 \leq k \leq n$:

$$M\varphi_k = \lambda_k \varphi_k \quad \textit{und} \quad \psi_k^\intercal M = \lambda_k \psi_k^\intercal.$$

(3) Für den Spektralradius von M gilt: $\rho(M) = 1$.

Beweis. (1): Zunächst halten wir fest, dass nach Definition von M und L_{rw} gilt, dass $M = I_d - L_{rw}$. Sei nun φ ein Eigenvektor von L_{rw} zum Eigenwert λ. Dann gilt

$$M\varphi = (I_d - L_{rw})\,\varphi = \varphi - \lambda\varphi = (1-\lambda)\varphi,$$

was zu zeigen war. Insbesondere entsprechen die k größten Eigenvektoren von M den k kleinsten Eigenvektoren von L_{rw}, da alle Eigenwerte von L_{rw} nach Proposition 3.4 nicht negativ sind.

(2): Betrachten wir die Matrix $S = D^{\frac{1}{2}} M D^{-\frac{1}{2}} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$. Diese ist nach Definition genau wie L_{sym} aus Kapitel 3 symmetrisch ist und lässt sich deshalb mit dem Spektralsatz in

$$S = V \Lambda V^\intercal$$

zerlegen, wobei Λ die Matrix mit den absteigenden Eigenwerten von S auf der Hauptdiagonalen ist und $V = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}$ die Eigenvektoren von S spaltenweise enthält. Diese bilden nach geeigneter Normalisierung eine Orthonormalbasis des $\mathbb{R}^n$, also gilt $V^\intercal V = I_d$.
Damit folgt:

$$M = D^{-\frac{1}{2}} S D^{\frac{1}{2}} = D^{-\frac{1}{2}} V \Lambda V^\intercal D^{\frac{1}{2}} = \left(D^{-\frac{1}{2}} V\right) \Lambda \left(D^{\frac{1}{2}} V\right)^\intercal.$$

Definieren wir nun $\Phi = D^{-\frac{1}{2}} V$ mit $\Phi = \begin{pmatrix} \varphi_1 & \cdots & \varphi_n \end{pmatrix}$, $\varphi_k = D^{-\frac{1}{2}} v_k$ und $\Psi = D^{\frac{1}{2}} V$ mit $\Psi = \begin{pmatrix} \psi_1 & \cdots & \psi_n \end{pmatrix}$, $\psi_k = D^{\frac{1}{2}} v_k$, dann gilt

$$M = \Phi \Lambda \Psi^\intercal,$$

wobei $\Phi^\intercal \Psi = \left(D^{-\frac{1}{2}} V\right)^\intercal D^{\frac{1}{2}} V = V^\intercal V = I_d$.
Nun gilt mit $V^\intercal V = I_d$, dass

$$\begin{aligned}\Psi^\intercal M &= \left(D^{\frac{1}{2}} V\right)^\intercal \left(D^{-\frac{1}{2}} V\right) \Lambda \left(D^{\frac{1}{2}} V\right)^\intercal = V^\intercal D^{\frac{1}{2}} D^{-\frac{1}{2}} V \Lambda \left(D^{\frac{1}{2}} V\right)^\intercal \\ &= \Lambda \left(D^{\frac{1}{2}} V\right)^\intercal = \Lambda \Psi^\intercal.\end{aligned}$$

Also ist Ψ die Matrix der Linkseigenvektoren von M als Spalten. Durch Multiplikation der Matrix Φ von rechts an M, folgt analog, dass Φ die Matrix der Rechtseigenvektoren von M als Spalten ist, was zu zeigen war.

(3): Sei nun λ ein Eigenwert mit zugehörigem Rechtseigenvektor φ von M und $1 \leq i_{max} \leq n$ diejenige natürliche Zahl, für die $|\varphi_{i_{max}}|$ der betragsmäßig größte Eintrag von φ ist. Insbesondere ist $|\varphi_{i_{max}}| > 0$, da φ sonst der Nullvektor wäre. Wir erhalten dann zum einen

$$\lambda \, \varphi_{i_{max}} = (M\varphi)_{imax} = \sum_{j=1}^{n} M_{imax,j} \, \varphi_j$$

und mit Division durch $\varphi_{i_{max}}$ sowie der Dreiecksungleichung außerdem

$$|\lambda| = \left| \sum_{j=1}^{n} M_{imax,j} \, \frac{\varphi_j}{\varphi_{i_{max}}} \right| \leq \sum_{j=1}^{n} |M_{imax,j}| \, \left| \frac{\varphi_j}{\varphi_{i_{max}}} \right| \leq \sum_{j=1}^{n} |M_{imax,j}| = 1.$$

Im letzten Schritt haben wir zudem genutzt, dass M eine zeilenstochastische Übergangsmatrix ist, deren Zeilensummen Eins betragen.

□

Die Zerlegung $M = \Phi \Lambda \Psi^\intercal$ aus obiger Proposition können wir aufgrund de Diagonalität von Λ umschreiben zu

$$M = \sum_{k=1}^{n} \lambda_k \varphi_k \psi_k^\intercal. \tag{5.4}$$

Da wir zudem wissen, dass $\Phi^\intercal \Psi = I_d$, gilt außerdem

$$M^2 = (\Phi \Lambda \Psi^\intercal)^2 = \Phi \Lambda \Psi^\intercal \Phi \Lambda \Psi^\intercal = \Phi \Lambda^2 \Psi^\intercal$$

und induktiv für beliebiges $t \in \mathbb{N}$

$$M^t = \Phi \Lambda^t \Psi^\intercal.$$

Damit folgt wie in (5.4)

$$M^t = \sum_{k=1}^{n} \lambda_k^t \varphi_k \psi_k^\intercal. \tag{5.5}$$

Betrachten wir nun den bereits angesprochenen Ansatz einem Knoten v_i, der Wahrscheinlichkeitsverteilung einer Irrfahrt mit Start in v_i nach t Schritten zuzuordnen, also

$$v_i \to M^t[i,:] = \sum_{k=1}^{n} \lambda_k^t \, \varphi_{i,k} \, \psi_k^\intercal,$$

wobei $\varphi_{i,k}$ den i-ten Eintrag des Eigenvektors φ_k darstellt.
Diffusionsabbildungen bestehen nun im Wesentlichen aus den Koordinaten dieser Abbildung bezüglich der Basis $\{\psi_k^\intercal\}_{k=1,\dots,n}$. Die Koordinate bezüglich $\psi_1^\intercal$ lassen wir dabei weg, da $\varphi_1 = \mathbf{1}$ und somit die erste Koordinate für jeden Knoten gleich ist. Sie trägt deshalb nicht dazu bei, zwischen verschiedenen Knoten zu unterscheiden [2, S.68].

Definition 5.2. *Sei $G = (V, E, W)$ ein gewichteter Graph und $M = \Phi\Lambda\Psi$ wie in (5.3) zerlegt.*
Dann ist die Diffusionsabbildung $\phi_t : V \to \mathbb{R}^{n-1}$ gegeben durch

$$\phi_t(v_i) := \begin{pmatrix} \lambda_2^t \, \varphi_{i,2} \\ \lambda_3^t \, \varphi_{i,3} \\ \vdots \\ \lambda_n^t \, \varphi_{i,n} \end{pmatrix}. \tag{5.6}$$

Für zwei Knoten v_i und v_j unseres Graphen bezeichnen wir zudem

$$\|\phi_t(v_i) - \phi_t(v_j)\|_2 \tag{5.7}$$

als den Diffussionsabstand der beiden Knoten nach t Zeitschritten.

Man sieht, dass jede Koordinate durch λ_k^t skalliert ist. Wir wissen nach Proposition 5.1, dass $|\lambda_k| \leq 1$. Somit wird, falls λ_k klein ist, λ_k^t für moderates t erst recht klein, was folgende Definition motiviert [2, S. 68].

Definition 5.3 (Gekürzte Diffusionsabbildung)**.** *Gegeben sei $G = (V, E, W)$ ein gewichteter Graph, eine Dimension $d \leq n$ und $M = \Phi\Lambda\Psi$ wie in* (5.3) *zerlegt. Dann ist die zur Dimension d gekürzte Diffusionsabbildung $\phi_t^{(d)} : V \to \mathbb{R}^d$ gegeben durch*

$$\phi_t^{(d)}(v_i) := \begin{pmatrix} \lambda_2^t \, \varphi_{i,2} \\ \lambda_3^t \, \varphi_{i,3} \\ \vdots \\ \lambda_{d+1}^t \, \varphi_{i,d+1} \end{pmatrix}. \tag{5.8}$$

Mit dieser Definition erhalten wir zum einen für eine Menge von Daten $\{x_1, \ldots, x_n\}$ aus dem $\mathbb{R}^n$ eine Möglichkeit zur Dimensionsreduzierung, indem wir die Daten zunächst mittels der Gaußschen Ähnlichkeitsfunktion aus (2.3) in einen voll verbundenen gewichteten Graphen umwandeln und die Knoten dieses Graphen mittels einer d-dimensionalen Diffusionsabbildung in den $\mathbb{R}^d$ abbilden. Andererseits erhalten wir auch die Möglichkeit, Daten in den $\mathbb{R}^d$ einzubetten, die schon in Form eines Graphen gegeben sind.

Nun wollen wir zeigen, dass der in (5.7) definierte Diffusionsabstand, also die Euklidische Distanz in Diffusionsabbildungskoordinaten in sinnvoller Art und Weise den Abstand zweier Knoten misst. Nachfolgender Satz zeigt, dass der Diffusionsabstand die Distanz zweier Knoten v_{i_1} und v_{i_2} misst, indem er für alle Knoten v_j die Differenzen der Wahrscheinlichkeiten, dass zwei Irrfahrten startend in Knoten v_{i_1} und v_{i_2} nach t Iterationen auf Knoten v_j landen, aufsummiert und die jeweiligen Differenzen dabei mit den Kehrwerten der Knotengrade d_j gewichtet. Dadurch werden die Wahrscheinlichkeitsdifferenzen für Knoten mit geringerem Knotengrad stärker gewichtet als für Knoten mit höherem Grad.

Satz 5.4 ([2, Theorem 5.6])**.** *Für alle Paare von Knoten v_{i_1}, v_{i_2} von $G = (V, E, W)$ gilt*

$$\|\phi_t(v_{i_1}) - \phi_t(v_{i_2})\|_2^2 = \sum_{j=1}^{n} \frac{1}{d_j} \left[P(X_t = v_j \mid X_0 = v_{i_1}) \; - \; P(X_t = v_j \mid X_0 = v_{i_2})\right]^2.$$

Beweis. Nach (5.5) wissen wir, dass

$$M^t = \sum_{k=1}^{n} \lambda_k^t \varphi_k \psi_k^{\mathsf{T}}.$$

Also gilt für einen Eintrag von M^t

$$M_{ij}^t = \sum_{k=1}^{n} \lambda_k^t \, \varphi_{i,k} \, \psi_{j,k}. \tag{5.9}$$

Mit dieser Vorüberlegung folgt

$$\begin{aligned}
&\sum_{j=1}^{n} \frac{1}{d_j} \left[P(X_t = v_j \mid X_0 = v_{i_1}) \; - \; P(X_t = v_j \mid X_0 = v_{i_2})\right]^2 \\
&= \sum_{j=1}^{n} \frac{1}{d_j} \left[M_{i_1 j}^t - M_{i_2 j}^t\right]^2 \overset{(5.9)}{=} \sum_{j=1}^{n} \frac{1}{d_j} \left[\sum_{k=1}^{n} \lambda_k^t \, \varphi_{i_1,k} \, \psi_{j,k} \; - \; \sum_{k=1}^{n} \lambda_k^t \, \varphi_{i_2,k} \, \psi_{j,k}\right]^2 \\
&= \sum_{j=1}^{n} \frac{1}{d_j} \left[\sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \, \psi_{j,k}\right]^2 = \sum_{j=1}^{n} \left[\sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \frac{\psi_{j,k}}{\sqrt{d_j}}\right]^2 \\
&= \sum_{j=1}^{n} \left[\sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \left(D^{-\frac{1}{2}} \psi_k\right)_j\right]^2 = \left\| \sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) D^{-\frac{1}{2}} \psi_k \right\|_2^2.
\end{aligned}$$

Da die Menge $\{D^{-\frac{1}{2}} \psi_k\}_{k=1,\ldots,n} = \{v_k\}_{k=1,\ldots,n}$ die Eigenvektoren der symmetrischen Matrix S aus Beweis von Proposition 5.1 darstellt, bilden diese eine Orthonormalbasis. Deshalb gilt für beliebige $a_k \in \mathbb{R}$ mit der Bilinearität des Skalarprodukts:

$$\begin{aligned}
&\left\langle \sum_{k=1}^{n} a_k v_k \, , \, \sum_{k=1}^{n} a_k v_k \right\rangle = \sum_{k=1}^{n} \left\langle a_k v_k \, , \, \sum_{k=1}^{n} a_k v_k \right\rangle = \sum_{k=1}^{n} \sum_{l=1}^{n} \langle a_k v_k \, , \, a_l v_l \rangle \\
&= \sum_{k=1}^{n} \langle a_k v_k \, , \, a_k v_k \rangle = \sum_{k=1}^{n} a_k^2 \, \langle v_k \, , \, v_k \rangle = \sum_{k=1}^{n} a_k^2.
\end{aligned} \tag{5.10}$$

Damit erhalten wir

$$\begin{aligned}
&\left\| \sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k})\, D^{-\frac{1}{2}} \psi_k \right\|_2^2 = \left\| \sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k})\, v_k \right\|_2^2 \\
&= \left\langle \sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k})\, v_k \,,\, \sum_{k=1}^{n} \lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k})\, v_k \right\rangle \\
&\overset{(5.10)}{=} \sum_{k=1}^{n} \left[\lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \right]^2 \langle v_k, v_k \rangle = \sum_{k=1}^{n} \left[\lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \right]^2 \\
&= \sum_{k=2}^{n} \left[\lambda_k^t (\varphi_{i_1,k} - \varphi_{i_2,k}) \right]^2 = \| \phi_t(v_{i_1}) - \phi_t(v_{i_2}) \|_2^2 \,,
\end{aligned}$$

wobei wir genutzt haben, dass $\varphi_1 = \mathbf{1}$, weshalb sich der Term für $k = 1$ wegkürzt. □

5.2. Zusammenhang zum Spectralen Clustering

Sowohl beim Spectralen Clustering als auch bei der Dimensionsreduzierung durch Diffusionsabbildungen benutzen wir die Eigenwerte und Eigenvektoren der zwei fast identischen Matrizen L_{rw} (beim Spectralen Clustering) und M (bei Diffusionsabbildungen), wobei die Eigenvektoren beider Matrizen nach Proposition 5.1 faktisch die gleichen sind. Mit folgendem Algorithmus, angelehnt an [2, Algorithmus 5.1], präzisieren wir diesen Zusammenhang, indem wir Spectrales Clustering mittels Diffusionsabbildungen beschreiben.

Algorithmus 5.5 (Spectrales Clustering beschrieben durch Diffusionsabbildungen)**.** *Eingabe: Ähnlichkeitsmatrix $S \in \mathbb{R}^{nxn}$, Anzahl k der zu findenden Cluster und Zeitschritt t.*

1. *Konstruiere zunächst einen zusammenhängenden Ähnlichkeitsgraphen durch einer der in Abschnitte 2.3 beschriebenen Methoden. Sei W die Gewichtsmatrix des Graphen.*
2. *Berechne die $M = D^{-1}W$.*
3. *Berechne für jeden Knoten v_i die $k-1$-dimensionale Diffusionsabbildung*

$$\phi_t^{(k-1)}(v_i) := \begin{pmatrix} \lambda_2^t \varphi_{i,2} \\ \lambda_3^t \varphi_{i,3} \\ \vdots \\ \lambda_k^t \varphi_{i,k} \end{pmatrix}$$

   mittels der k größten Eigenvektoren von M.
4. *Cluster die Punkte $\{\phi_t^{(k-1)}(v_1), \ldots, \phi_t^{(k-1)}(v_n)\}$ in $\mathbb{R}^{k-1}$ mittels des k-means Algorithmus in Cluster $C_1, \ldots, C_k$.*
5. *Erhalte Knotenpartition $A_1, \ldots, A_k$, wobei $A_i = \{v_j | \phi_t^{(k-1)}(v_j) \in C_i\}$.*

Betrachten wir für Algorithmus 5.5 den Fall $t = 0$, das heißt, wir ignorieren die Skallierungen mit den Potenzen λ_k^t der Eigenwerte. Wir wollen nun zeigen, dass in diesem Fall Algorithmus 5.5 genau dem anfangs vorgestellten Algorithmus 3.6 entspricht, wenn wir dort nur die ersten $k-1$ nicht trivialen Eigenvektoren betrachten (s. Bemerkung 3.7). Zur Erinnerung: der Eigenvektor, der zum kleinsten Eigenwert von L_{rw} gehört, ist ein Vielfaches des konstanten Vektors $\mathbf{1}$.

Wir haben bereits in Proposition 5.1 gesehen, dass die k kleinsten Eigenvektoren $u_1, \ldots, u_k$ mit zugehörigen Eigenwerten $0 = \lambda_1 \leq \cdots \leq \lambda_k$ von L_{rw} genau den k größten Eigenvekoren $\varphi_1, \ldots, \varphi_k$ von M mit Eigenwerten $1 = 1 - \lambda_1 \geq \cdots \geq 1 - \lambda_k$ entsprechen. Wenn wir nun in Algorithmus 5.5 für $t = 0$ die Punkte

$\{\phi_0^{(k-1)}(v_1), \ldots, \phi_0^{(k-1)}(v_n)\}$ mittels k-means clustern, gilt für einen dieser Punkte

$$\phi_0^{(k-1)}(v_i) = \begin{pmatrix} \lambda_2^0 \, \varphi_{i,2} \\ \lambda_3^0 \, \varphi_{i,3} \\ \vdots \\ \lambda_k^0 \, \varphi_{i,k} \end{pmatrix} = \begin{pmatrix} \varphi_{i,2} \\ \varphi_{i,3} \\ \vdots \\ \varphi_{i,k} \end{pmatrix} = \begin{pmatrix} u_{i,2} \\ u_{i,3} \\ \vdots \\ u_{i,k} \end{pmatrix} = y_i$$

wobei y_i die i-te Zeile der Matrix U aus Algorithmus 3.6 ist, dessen Spalten die Eigenvektoren $u_2, \ldots, u_k$ sind, wenn wir den ersten Eigenvektor $u_1 = \mathbf{1}$ nicht beachten. Wir wenden also k-means in beiden Fällen auf genau die gleichen Punkte an.

Dementsprechend kann Spectrales Clustering tatsächlich als einfaches Anwenden von k-means auf die Einbettung unserer Daten, die durch die auf $k-1$ Dimensionen gekürzte Diffusionsabbildung für $t = 0$ gegeben ist, verstanden werden. Wir umgehen dabei durch die veränderte Repräsentation der Daten oftmals typische Probleme von k-means beispielsweise beim Umgang mit nicht konvexen Clustern.

6. Anwendungen

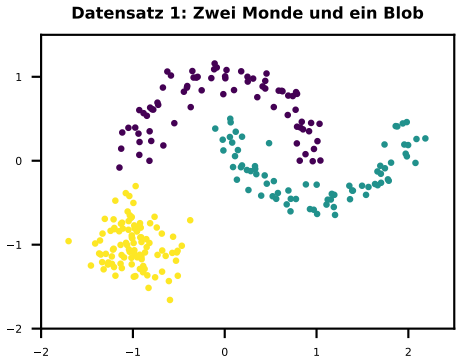
In diesem Abschnitt wollen wir uns mit den Problemen beschäftigen, die bei der tatsächlichen Implementierung und Verwendung von Spectralem Clustering auftreten. Insbesondere die Konstruktion eines Ähnlichkeitsgraphen aus gegebenen Daten ist eine nicht triviale Aufgabe, bei der teilweise mehrere Parameter gesetzt werden müssen. Wir betrachten dazu in diesem Abschnitt Spielzeugbeispiele (s. Abbildung 4), an denen wir die Wahl des Ähnlichkeitsgraphen und deren Parameter diskutieren. Am Schluss werden wir unsere Erkenntnisse noch auf den recht bekannten Iris-Datensatz von Ronald Fisher [9] anwenden. Theoretische Erkenntnisse zur Konstruktion des Ähnlichkeitsgraphen gibt es kaum [19, S. 17].

6.1. Wahl der Ähnlichkeitsfunktion

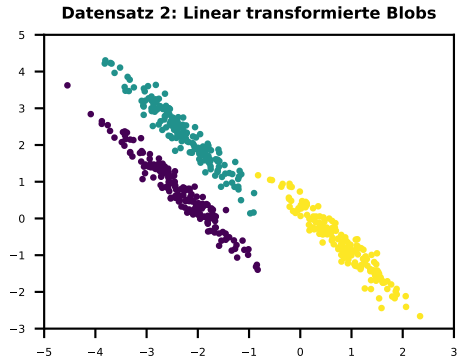
Bevor wir einen Ähnlichkeitsgraphen erstellen können, müssen wir zunächst eine Ähnlichkeitsfunktion definieren, die die lokalen Nachbarschaftsbeziehungen unserer Daten aussagekräftig abbildet. Dabei ist das globale Langzeitverhalten der Funktion wenig relevant. Beispielsweise ist es für zwei Punkte x_i und x_j, die weit voneinander entfernt liegen, egal, ob diese einen Ähnlichkeitswert s_{ij} von $0,001$ oder $0,01$ haben [19, S.17]. Beim kNN Graphen und beim ϵ-Nachbarschaftsgraphen würden diese Punkte ziemlich ohnehin sowieso nicht verbunden werden. Da die Beispiele aus dieser Arbeit aus dem $\mathbb{R}^p$ stammen, bertachten wir als Ähnlichkeitsfunktion einen Standardkandidaten, die bereits vorgestellte Gaußsche Ähnlichkeitsfunktion

$$s(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}\right) = \exp\left(\frac{-{d_{ij}}^2}{2\sigma^2}\right)$$

aus (2.3).Die Wahl des Parameters σ werden wir Laufe dieses Abschnitts noch diskutieren.



(a) Punkte $X_1, \ldots, X_{150}$ aufgeteilt in zwei Monde mit $\sigma^2 = 0.1^2$ und $X_{151}, \ldots, X_{250} \sim \mathcal{N}_2((-1\ -1)^\intercal, \Sigma)$ mit $\Sigma = 0.25^2 I_d$.



(b) Punkte $X_1, \ldots, X_{500}$ mit $X_i = AY_i$ mit $A = \begin{pmatrix} 0.5 & -0.4 \\ -0.5 & 0.7 \end{pmatrix}$ und $Y_1, \ldots, Y_{167} \sim \mathcal{N}_2((-8.95\ -5.46)^\intercal, I_d)$, $Y_{168}, \ldots, Y_{334} \sim \mathcal{N}_2((-4.59\ 0.09)^\intercal, I_d)$, $Y_{335}, \ldots, Y_{500} \sim \mathcal{N}_2((1.94\ 0.51)^\intercal, I_d)$.

Abb. 4.: Zwei Beispieldatensätze: Der erste Datensatz besteht aus zwei Monden und einem Blob. Als Monde bezeichnen wir zwei Punktmengen, bei denen die Punkte zunächst auf zwei ineinanderliegenden Halbkreisen liegen und dann für jeden dieser Punkte in beiden Dimensionen eine normalverteilte Zufallsvariable (ZV) mit Mittelwert $\mu = 0$ und Varianz σ^2 aufaddiert wird. Als Blob bezeichnen wir eine Menge von zweidimensional Normalverteilten ZVn mit Erwartungswertvektor $\mu \in \mathbb{R}^2$ und Kovarianzmatrix $\Sigma \in \mathbb{R}^{2x2}$. Der zweite Datensatz besteht grundlegend aus 500 Punkten aufgeteilt in drei Blobs. Jeder Punkt wurde dann zusätzlich mittels einer linearen Abbildung verschoben.

6.2. Wahl des Ähnlichkeitsgraphen mit entsprechenden Parametern

Nun wollen wir das Verhalten der verschiedenen Ähnlichkeitsgraphen an unseren Beispieldatensätzen illustrieren. Bei der Erzeugung dieser haben wir die in Python-Bibliothek Scikit-learn [15] benutzt, aus der ebenfalls der von uns zur Auswertung verwendete Spectrale Clusteralgorithmus stammt. Bei der Analyse und graphischen Darstellung haben wir außerdem die Bibliotheken Numpy [5], Matplotlib [12] und Networkx [11] genutzt.

Bei der Erstellung des Ähnlichkeitsgraphen sollten wir immer beachten, dass wenn der Graph aus mehreren Zusammenhangskomponenten besteht, die Eigenvektoren von L_{rw} Indikatorvektoren sind (Proposition 3.5), durch die k-means die Cluster trivial erkennt [19, S. 18]. Dies kann dann von Vorteil sein, wenn man sich sicher ist, dass die Komponenten den zu findenden Clustern entsprechen. Da man beim Clustering aber im Normalfall über kein a priori Wissen über die Daten verfügt, ist dies im Allgemeinen nicht der Fall. Deshalb ist es in der Regel sinnvoll, den Ähnlichkeitsgraphen so zu wählen, dass dieser verbunden ist oder zumindest aus deutlich weniger Komponenten besteht, als wir Cluster finden wollen.

Der ϵ-Nachbarschaftsgraph (ϵ-Graph) ist dann gut geeignet, wenn unsere Daten überall ähnlich skalliert sind, also die Distanz zwischen Daten in verschiedenen Bereichen ungefähr gleich ist ([19, S. 17]). Ist dies allerdings nicht der Fall, können wir mit diesem Graphen Probleme bekommen. Betrachten wir dazu den ϵ-Graph für $\epsilon = 0.25$ aus Abbildung 5. Für diesen Wert zerfällt der Graph in mehrere Komponenten und einzelne Knoten, die gar nicht verbunden sind. Dementsprechend funktioniert hier auch das Spectrale Clustering nicht wie erhofft. Von Luxburg schlägt in [19] deshalb folgendes Verfahren vor, wie wir gewährleisten können, dass der Graph sicher verbunden ist. Das Verfahren bezeichnen wir ab jetzt als „MST-Methode“. Dabei erstellen wir aus unseren Daten zunächst einen voll verbundenen Graphen, mit paarweisen Distanzen als Gewichte der Kanten. Für diesen Graphen berechnen wir dann beispielsweise mit dem Algorithmus von Kruskal einen minimalen Spannbaum (engl. minimum spanning tree (MST)). Für genauere Informationen zu minimalen Spannbäumen verweisen wir auf [6, Seite 635 ff.]. Schließlich wählen wir ϵ als längste Kanten des MST.

Verwenden wir diese Methode für den ersten Beispieldatensatz, sehen wir ebenfalls in Abbildung 5, dass der Graph nun verbunden ist und wir ein gutes Clustering erhalten. Allerdings kann diese Heuristik auch dazu führen, dass wir ϵ zu groß wählen, um die wahre Nachbarschaftsbeziehungen der Punkte widerzuspiegeln. Betrachten wir dazu die Anwendung des ϵ-Graphen auf Datensatz 2 in Abbildung 6. Hier führt die MST-Methode dazu, dass selbst der Ausreißerpunkt oben links mit verbunden wird. Das hat zur Folge, dass die beiden oberen Punktmengen zu stark miteinander verbunden werden. Da wir den ϵ-Graph meist ungewichtet betrachten, scheitert für diesen Graphen das Spectrale Clustering. Mit $\epsilon = 0.5$

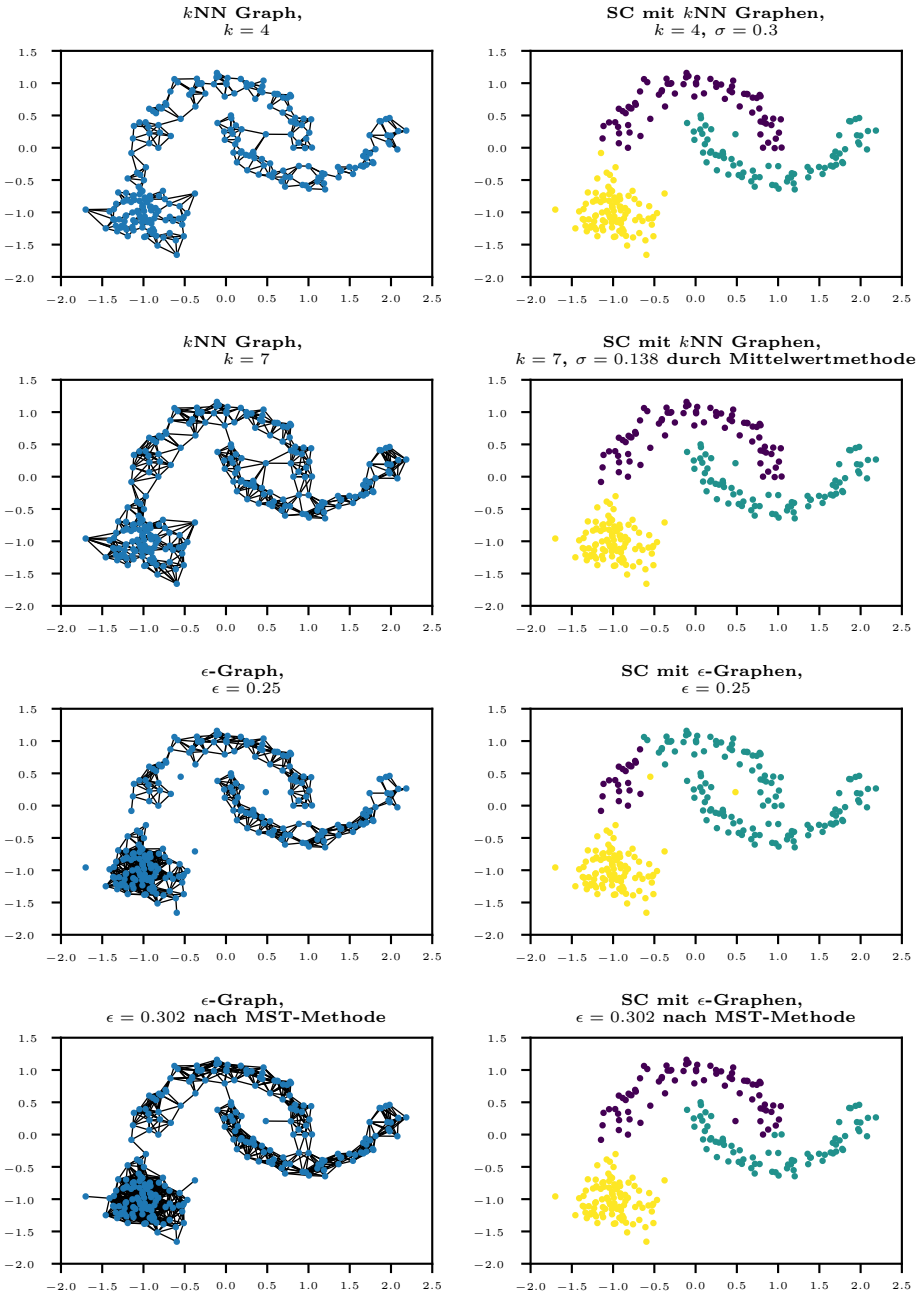


Abb. 5.: kNN- und ϵ-Graph mit entsprechendem Spectralem Clustering für Beispieldatensatz 1.

hingegen schaffen wir es zwar nicht den Graphen zusammenhängend zu wählen (der Ausreißerpunkt oben links ist nicht verbunden), jedoch werden dadurch die beiden oberen Punktmengen nicht allzu stark verbunden. Das Spectrale Clustering liefert, da der restliche Graph ansonsten zusammenhängend ist, entsprechend ein gutes Ergebnis.

Der kNN-Graph wiederum hat den Vorteil auch Punkte aus Bereichen mit verschiedener Dichte zu verbinden zu können. Von Luxburg erwähnt in [19] die theoretische Erkenntnis, dass für den asymptotischen Fall $n \to \infty$ der Graph mit hoher Wahrscheinlichkeit verbunden ist, wenn wir $k \sim log(n)$ wählen. Aus diesem Grund rät von Luxburg, für große Datensätze $k \sim log(n)$ zu wählen und für kleinere Datensätze verschiedene Werte für k auszuprobieren. Für unsere Beispieldatensätze bestehend aus 250 und 500 Punkten haben wir jeweils für $k = \lceil \log(n) + 1 \rceil$ und verschiedene Werte von σ sehenswerte Resultate erzielt (s. Abbildug 5 und 6).

Zu erwähnen ist noch, dass je größer wir k-wählen, desto mehr Knoten, die weiter voneinander entfernt sind, werden miteinander verbunden und desto kleiner sollten wir σ der Ähnlichkeitsfunktion wählen, um den resultierenden Kanten ein nicht zu großes Gewicht zuzuweisen. Im Allgemeinen ist der kNN-Graph aber flexibel in der Paramterwahl, solange man k groß genug wählt, sodass der Graph zusammenhängend ist. Wählt man beispielsweise für den zweiten Datensatz $k = 3$, so zerfällt der entsprechende kNN-Graph in mehrere Komponenten und das Spectrale Clustering scheitert (Abbildung 6).

Beim **voll verbundenen Graphen** muss zunächst der Nachteil genannt werden, dass die aus diesem Graphen resultierende Ähnlichkeitsmatrix, anders als beim ϵ- oder kNN-Graphen, nicht dünn besetzt ist. Dies kann, besonders auf große Datensätze angewandt, zu numerischen Problemen bei der Berechnung der Eigenwerte führen.
Falls man sich dennoch für den voll-verbundenen Graphen entscheidet, ist es wünschenswert, dass der Graph ähnliche Eigenschaften besitzt, wie sie auch ein kNN- oder ϵ-Graph haben würde [19, S. 19]. Die Kunst liegt hier in der Wahl eines passenden σ. Betrachten wir die Irrfahrt auf dem voll verbundenen Graphen mit Gewichten durch die Gaußsche Ähnlichkeitsfunktion. Dann führt eine Verringerung

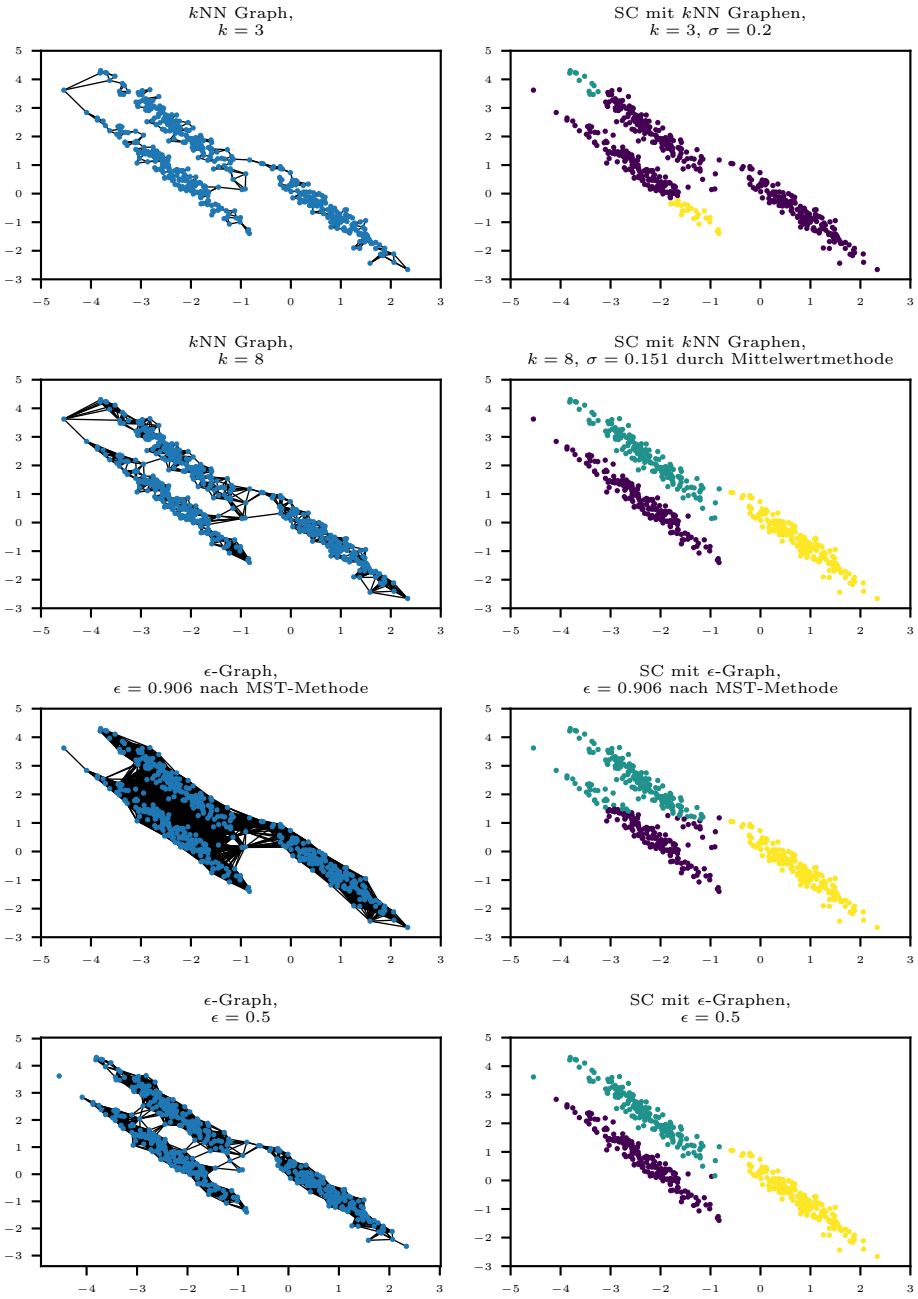


Abb. 6.: kNN- und ϵ-Graph mit entsprechendem Spectralem Clustering für Beispieldatensatz 2.

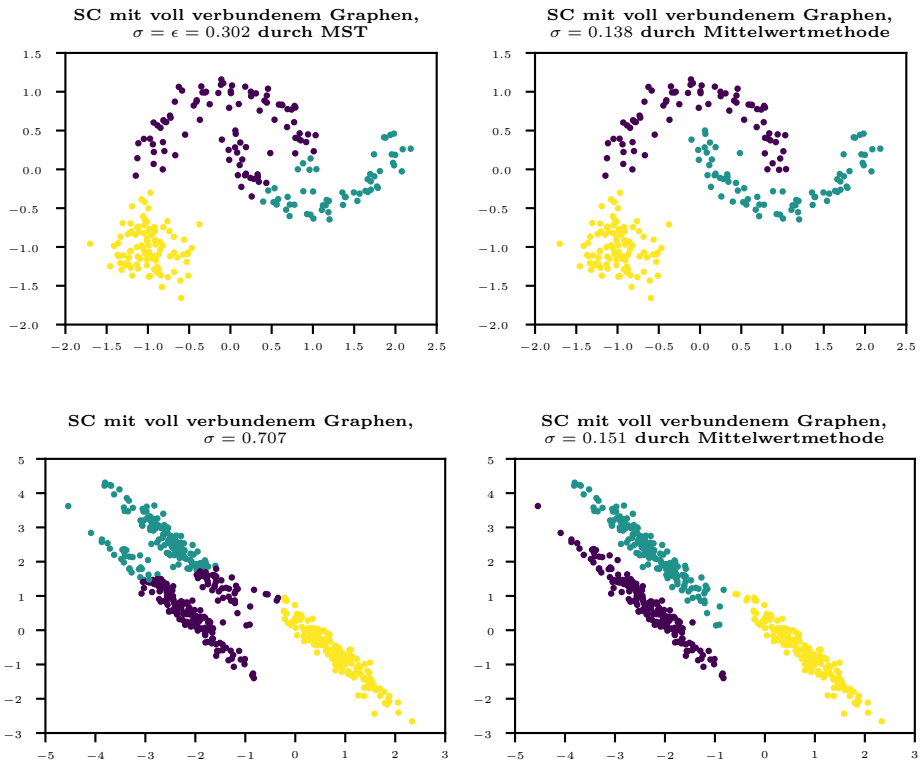


Abb. 7.: Spectrales Clustering mit voll verbundenem Graphen für Beispieldatensätze.

von σ dazu, dass die Wahrscheinlichkeit, dass die Irrfahrt einen Häufungsbereich verlässt, schrumpft. Eine Erhöhung von σ führt analog zu einer Erhöhung dieser Wahrscheinlichkeit.
Von Luxburg schlägt in [19, S. 19] zwei Heuristiken vor, σ zu wählen. Ein erster Ansatz ist es, ϵ durch die MST-Methode zu bestimmen und $\sigma = \epsilon$ zu wählen. Dieser Ansatz hat jedoch bei uns weder für den ersten noch für den zweiten Datensatz zu vielversprechenden Clusterergebnissen geführt (s. Abbildung 7).

Ein zweiter Ansatz für die Wahl von σ ist folgende Heuristik, die wir fortan als Mittelwertmethode bezeichnen. Zunächst wählen wir $k \sim log(n) + 1$ (konkret wählen wir $k = \lceil \log(n) + 1 \rceil$). Als nächstes berechnen wir den mittleren Abstand jedes Punktes zu seinen k nächsten Nachbarn, um schließlich σ als den Mittelwert

dieser Abstände zu wählen. Diese Methode führte in unseren Analysen für den voll verbundenen Graphen bei beiden Datensätzen zu vielversprechenden Ergebnissen. Dies motivierte uns deshalb $k = \lceil \log(n) + 1 \rceil$ und σ durch die Mittelwertmethode ebenfalls für den kNN-Graphen auszuprobieren. Insbesondere mit dieser Wahl der Parameter lieferte der kNN-Graph für beide Beispieldatensätze sehr gute Ergebnisse, was uns dazu animierte, den kNN-Graphen mit Parametern k und σ aus der Mittelwertmethode auf einen echten Datensatz anzuwenden.
Wir betrachteten deshalb den bekannten Iris-Datensatz von Ronald Fisher (s. [9]). Dieser vierdimensionale Datensatz besteht aus jeweils 150 Datenpunkten, wobei jeweils 50 Datenpunkte einer von drei verschiedenen Iris-Spezies (Iris setosa, Iris virginica und Iris versicolor) angehören. Wie man in Abbildung 8 erkennen kann, konnten alle Pflanzen der setosa Spezies korrekt zugeordnet werden. Die Pflanzen der virginica-Spezies ordneten wir zwar einem Cluster zu, allerdings gruppierten wir in diesem Cluster ebenfalls 14 Pflanzen der Gattung versicolor ein. Alles in allem konnten wir 136 der 150 Pflanzen richtig zuordnen, ein zufriedenstellendes Ergebnis (zum Vergleich: mit k-means konnten wir nur 134 Pflanzen richtig zuordnen).

	Cluster 0	Cluster 1	Cluster 2
setosa	50	0	0
versicolor	0	50	0
virginica	0	14	36

Abb. 8.: Ergebnisse des Spectrales Clusterings mit kNN-Graphen und Werten aus Mittelwertmethode für Iris Datensatz.

Als Faustregel empfehlen wir deshalb, bei der Wahl des Ähnlichkeitsgraphen zunächst auf den kNN-Graphen mit Parametern der Mittelwertmethode zu setzen. Dieser Graph liefert für verschiedene Werte von k und σ gute Lösungen, führt immer zu einer dünnbesetzten Ähnlichkeitsmatrix und war am wenigsten anfällig für ungeeignete Parameterwahlen. Für kleine bis mittelgroße Datensätze kann man

alternativ auch den voll verbundenen Graphen mit dem Vorteil wählen, nur einen Parameter festlegen zu müssen. Der ϵ-Graph führte bei uns zu den am wenigsten konstanten Lösungen. Wir würden diesen Graphen nur empfehlen, wenn bekannt ist, dass die Daten einigermaßen gleich skalliert sind oder wenn man den Graphen, anders als von von Luxburg in [19] vorgeschlagen, gewichtet betrachten möchte. In diesem Fall hätte der ϵ-Graph den Vorteil, dass man diesen durch die oben vorgestellte MST-Methode in jedem Fall zusammenhängend wählen könnte und dabei durch eine Gewichtung eine bessere Darstellung der lokalen Nachbarschaftsbeziehungen zwischen den Punkten erhält als durch einen ungewichteten ϵ-Graph. Im Großen und Ganzen sind dies aber nur Faustregeln, die auf keiner theoretischen Erkenntnis basieren.

7. Zusammenfassung und Ausblick

Wir haben in dieser Arbeit gesehen, dass Spectrales Clustering im Wesentlichen darauf basiert, Daten in einen Graphen umzuwandeln und für diesen mittels der ersten k Eigenvektoren der Laplace-Matrizen, die Minimierung des normalisierten Schnittes zu approximieren. Zudem wurde deutlich, dass die Eigenvektoren von Laplace-Matrizen ebenfalls eine entscheidende Rolle bei der Dimensionsreduktion durch Diffusionsabbildungen spielen. Wir konnten deshalb Spectrales Clustering außerdem dadurch interpretieren, dass wir unsere Daten durch Diffusionsabildungen in den $\mathbb{R}^{k-1}$ einbetten und auf die eingebetteten Daten einen einfachen Clusteralgorithmus, wie k-means, anwenden.

Bei der Anwendung von Spectralem Clustering auf Beispieldatensätze zeigte sich, dass Spectrales Clustering, anders als zum Beispiel k-means, keine Annahmen über die Form der Cluster macht und deshalb gut geeignet ist, um Datensätze beliebiger Formen zu gruppieren. Beispielsweise hatten wir nach Wahl eines passenden Ähnlichkeitsgraphen keine Probleme, zwei in sich verschachtelte mondförmige Punktmengen voneinander zu unterscheiden.
Hier zeigt sich jedoch auch eine Schwäche der Spectralen Clusteranalyse. Die Wahl eines geeigneten Ähnlichkeitsgraphen ist alles andere als eine triviale Aufgabe. Für unpassende Wahlen erhielten wir instabile Ergebnisse. Zwar konnten wir mit Hilfe von [19] für uns Faustregeln festlegen, die bei unseren Beispielen gut funktionierten, dennoch basierten diese nicht auf konkreten theoretischen Erkenntnissen. Spectrales Clustering ist dementsprechend kein Black-Box Algorithmus, der automatisch gute Ergebnisse liefert. Aus unserer Sicht stellt die Forschung nach theoretischen Erkenntnissen über die passende Wahl des Ähnlichkeitsgraphen mit entsprechenden Parametern deshalb ein spannendes Forschungsgebiet für die Zukunft dar. Bach und Jordan liefern in [1] bereits einen ersten Ansatz eines Algorithmus, aus Datensätzen mit bekannten Clustern für die Wahl der Ähnlichkeitsmatrix zukünftiger Datensätze zu lernen.

Auf den Aspekt der Wahl von der optimalen Anzahl k der zu findenden Cluster sind wir in dieser Arbeit nicht eingegangen. Dazu finden sich zum einen in [19] spannende Erkenntnisse, die die Eigenwerte der Laplace-Matrizen ausnutzen, zum anderen ist die „elbow method “, die ebenfalls bei der Dimensionreduktion mittels Hauptkomponentenanalyse genutzt wird, eine bedeutsame Heuristik ([2], [20]) für die Wahl eines geeigneten k.

Zusammengefasst haben wir gesehen, dass Spectrales Clustering richtig angewandt selbst für komplizierte und große Datensätze, sofern die entsprechende Ähnlichkeitsmarix dünn besetzt ist, mittels klassischer Methoden der Linearen Algebra genaue Lösungen findet. Wir sehen deshalb in der weiteren Forschung großes Potenzial für die Zukunft.

A. Literaturverzeichnis

[1] F. Bach and M. Jordan. Learning Spectral Clustering. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003.

[2] A. S. Bandeira, A. Singer, and T. Strohmer. Mathematics of Data Science (draft), 2020.

[3] A. S. Bandeira and N. Zhivotovskiy. Mathematics of Machine Learning: Lecture Notes, 2021.

[4] M. Birkner. Einführung in die Stochastik: Notizen zu einer Vorlesung an der Johannes-Gutenberg-Universität Mainz, Wintersemester 2020/2021. `https://www.staff.uni-mainz.de/birkner/GrundlStoch_2021/Stochastik-Einfuehrung_WS2021.pdf`.

[5] R. Charles et al. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Algorithmen - eine Einführung*. Oldenbourg Verlag, München, 4., durchgesehene und korrigierte Auflage edition, 2013.

[7] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

[8] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[9] R. A. Fischer. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[10] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley. Laplacian-based dimensionality reduction including spectral clustering, laplacian eigenmap, locality preserving projection, graph embedding, and diffusion map: Tutorial and survey.

[11] A. Hagberg et al. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[13] H. Lütkepohl. *Handbook of matrices*. Wiley, Chichester, 1996.

[14] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[15] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics*, 39(4):1878 – 1915, 2011.

[17] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[18] P. Sondergaard. Quote: Sondergaard on data analytics. `https://www.causeweb.org/cause/resources/library/r2493`, 2011. [eingesehen am 11.07.2022].

[19] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[20] Wikipedia contributors. Elbow method (clustering) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Elbow_method_(clustering)&oldid=1095735775`, 2022. [eingesehen am 01.07.2022].

[21] Wikipedia contributors. K-means clustering — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=K-means_clustering&oldid=1095144243`, 2022. [eingesehen am 01.06.2022].

[22] Wirtschaftsforum. Big data oder: Viel hilft viel?! https://www.wirtschaftsforum.de/expertenwissen/was-ist-denn-data-science/big-data-oder-viel-hilft-viel, 2022. [eingesehen am 10.07.2022].

B Programmcode

Im Folgenden repräsentieren wir den Code, den wir genutzt haben, um unsere Analysen und Grafiken zu erstellen. Dabei verzichten wir größtenteils auf den Code zur Erzeugung der Plots, da dieser immer gleich aufgebaut ist. Wir beschränken uns auf das Wesentliche.

[1]:
```
#Wir implementieren zunächst alle nötigen Bibliotheken
import numpy as np
from sklearn.cluster import SpectralClustering
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.neighbors import kneighbors_graph
from sklearn.neighbors import radius_neighbors_graph
from sklearn.metrics.pairwise import euclidean_distances
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib
```

Beispielgraphen mit Eigenvektoren: Abb. 2
Im Folgenden betrachten wir zwei einfache Beispiele für Graphen mit entsprechenden Eigenvektoren, die wir mit der Bibliothek networkx erzeugen. Zur Erinnerung ein Graph $G = (V, E)$ ist ein Tupel bestehend aus einer Knotenmenge V und einer Kantenmenge E, die angibt, welche Knoten miteinander verbunden sind. Die Graphen, die wir hier betrachten, sind ungewichtet.

[2]:
```
G_1 = nx.Graph()

#Knotenmenge
V_1 = np.array([1,2,3,4,5,6,7,8,9,10])
G_1.add_nodes_from(V_1)

#Kantenmenge
E_1 = [(1,2),(4,3),(3,2),(2,5),(5,1), (2,4),(4,5), (1,4), (6,7), (7,8),␣
 ↪(8,9), (9,10),
                    (10,6), (7,10), (6,9), (9,7), (8,3)] #
G_1.add_edges_from(E_1)

#Feste Positionen der Knoten für den Graphen, damit dieser beim plotten␣
 ↪immer gleich aussieht
pos_G = nx.spring_layout(G_1, seed=8)
nx.draw(G_1,pos = pos_G, with_labels = True)
```

Eine natürliche Art einen Graphen darzustellen ist über seine Adjazenzmatrix A, die durch

$$A_{ij} = \begin{cases} 1 & \text{für } \{v_i, v_j\} \in E \\ 0 & \text{sonst} \end{cases}$$

definiert ist. Die Adjazenzmatrix erhalten wir durch folgenden Befehl:

[3]:
```
A = nx.to_numpy_array(G_1)
```

Aus der Adjazenzmatrix wollen wir nun die normalisierte Laplace-Matrix $L_{rw} = I_d - D^{-1}W$ berechnen. Dazu berechnen wir zurnächst die Gradmatrix $D = diag(d_1, ..., d_n)$, wobei $d_i := \sum_{j=1}^{n} A_{ij}$. Nach Berechnung der beiden Matrizen berechnen wir die Eigenwerte und Eigenvektoren von L_{rw} und sortieren diese.

[ ]:
```
#Gradmatrix D
D = np.diag(A.sum(axis=1))

#Anzahl der Knoten im Graphen:
n_1 = A.shape[0]

#nxn Identitätsmatrix
Id = np.identity(D.shape[0])

#normalisierte Laplace-Matrix
L_rw = Id - np.matmul(np.linalg.inv(D), A) #Id - Matrixmultiplikation␣
↪D^{-1} * A

#Eigenwerte mit Eigenvektoren
Werte_1, Vek_1 = np.linalg.eig(L_rw)

#aufsteigende Sortierung nach den Eigenwerten
Vek_1 = Vek_1[:,np.argsort(Werte_1)]
Werte_1 = Werte_1[np.argsort(Werte_1)]

#Plotte Einträge der ersten 2 Eigenvektoren
fig, ax = plt.subplots(2)
fig.suptitle('Die ersten beiden EV von $L_{rw}$')

n = np.linspace(1, 100, 100 ) #natürliche Zahlen von 1 bis 100 (benötigen␣
↪wir für die Plots)
ax[0].plot(n[:n_1],Vek_1[:,0], 'r--o')
ax[0].grid()
ax[0].set_title(' $u_1 $')
ax[1].ticklabel_format(useOffset=False, style='plain')

ax[1].plot(n[:n_1],Vek_1[:,1], 'r--o')
ax[1].grid()
ax[1].set_title(' $u_2$ ')
ax[1].ticklabel_format(useOffset=False, style='plain')

fig.set_size_inches(5,4)
fig.tight_layout()
```

Man sieht, dass der erste Eigenvektor der konstante Vektor ist. Aus diesem erhalten wir also keine Informationen über den Graphen. Wenn wir den zweiten Eigenvektor betrachten,

sehen wir, dass die Einträge für $v_1, ..., v_5$ größer null und die Einträge für $v_6, ..., v_{10}$ kleiner null sind. Wenn wir unseren Graphen nun so aufteilen, dass wir in einem Cluster alle Knoten mit positivem Eintrag in u_2 und im anderen alle mit negativen Eintrag nähmen, wäre dies genau die intuitive Aufteilung, die wir erhielten, wenn wir den Graph zwischen Knoten v_3 und v_8 schneiden würden.

Im Folgenden verändern wir unseren Graphen etwas, indem wir eine Komponente (Haus des Nikolaus) hinzufügen. Dazu nehmen wir die Vereinigung des alten Graphen mit einem Haus des Nikolaus und fügen manuell eine Kanten hinzu.

[ ]:
```
G_2 = nx.Graph()
V_2 = np.array([11,12,13,14,15]) #Knotenmenge
G_2.add_nodes_from(V_2)

E_2 = [(11,12),(14,13),(13,12),(12,15),(15,11), (12,14),(14,15), (11,14)]␣
 ↪#Kantenmenge
G_2.add_edges_from(E_2)

#Vereinigung von G_1 und G_2
U = nx.union(G_1, G_2  )

#Füge manuell eine Kante hinzu
U.add_edges_from([(13, 3) ])

#Damit wir U jedes mal gleich zeichnen, legen wir auch hier die Position␣
 ↪für jeden Knoten fest
pos_U = nx.spring_layout(U, seed=4)
nx.draw(U, pos = pos_U, with_labels = True )
```

Wenn wir nun wieder die Eigenwerte und Eigenvektoren dieses Graphen berechnen und plotten erhalten wir Abbildung 2 aus dem Hauptteil der Arbeit.

Beispiel für Graphenschnitt (cut): Abb. 3
Wir wollen nun an folgendem Beispiel illustrieren, warum es beim Graphen-Clustering zu einfach ist, eine Partition A, A^c zu wählen, die

$$cut(A, A^c) = \sum_{i \in A, j \in A^c} w_{ij} \tag{1}$$

minimiert. Betrachten wir dazu folgenden gewichteten Graphen:

[ ]:
```
G = nx.Graph()
G.add_edge(0, 3, weight=0.25)

G.add_edge(1, 3, weight=0.25)
G.add_edge(1, 2, weight=0.3)
G.add_edge(2, 5, weight=0.25)
G.add_edge(2, 3, weight=0.3)
G.add_edge(3, 4, weight=0.3)
G.add_edge(3, 5, weight=0.2)
```

```
G.add_edge(4, 5, weight=0.5)
G.add_edge(4, 6, weight=0.15)
G.add_edge(4, 7, weight=0.15)
G.add_edge(6, 10, weight=0.4)
G.add_edge(6, 8, weight=0.4)
G.add_edge(6, 7, weight=0.4)
G.add_edge(7, 10, weight=0.3)
G.add_edge(8, 10, weight=0.3)
G.add_edge(8, 9, weight=0.4)
G.add_edge(9, 10, weight=0.3)
#Wir legen zum plotten feste Positionen fest
pos = nx.spring_layout(G, seed=70)
```

Wenn wir den Graphen mit dem Auge clustern würden, würde fast jeder vermutlich den Graphen intuitiv folgendermaßen teilen:

```
[ ]: nx.draw(G, pos, node_color=['r','r','r','r','r','r','y','y','y','y','y'],␣
      ↪node_size=300, with_labels = True)
     edge_labels= nx.draw_networkx_edge_labels(G,pos = pos, edge_labels=labels)
```

Wenn wir nun aber als Partition A, A^c der Knotenmenge V jene nehmen, welche $cut(A, A^c)$ minimiert, erhalten wir $A = \{0\}$ und $A^c = \{1, 2, ..., 10\}$ mit $cut(A, A^c) = 0.25$.

```
[ ]: nx.draw(G, pos, node_color=['r','y','y','y','y','y','y','y','y','y','y'],␣
      ↪node_size=300, with_labels = True)
     edge_labels= nx.draw_networkx_edge_labels(G,pos = pos, edge_labels=labels)
```

Der minimale Schnitt wird hier also durch die Abspaltung eines einzelnen Knoten erreicht. Dies ist natürlich nicht das Ergebnis, welches wir uns beim Clustern, bei dem wir einigermaßen ähnliche Gruppen identifizieren wollen, erhoffen. Deshalb betrachten wir in der Arbeit den normalisierten Schnitt. Die zweite hier erzeugten Grafik entspricht Abbildung 3 des Hauptteils.

Anwendungen von Spectralem Clustering auf Beispieldatensätze: Abb. 4-7
In diesem Abschnitt wollen wir untersuchen, wie der Wahl des Ähnlichkeitsgraphen mit entsprecheneden Parametern unser Ergebnis beim Spectralen Clustering beeinflusst. Dazu erzeugen wir 2 verschiedene Datensätze und betrachten verschiedene Ähnlichkeitsgraphen mit unterschiedlichen Parametern. Der k-means Algorithmus hat mit beiden Datensätze Probleme, wie wir sehen werden. Dazu betrachten wir zunächst einen ersten Beispieldatensatz.

Die Algorithmen, die wir betrachten werden, gruppieren Punkte, indem sie jedem Punkt X_i eine Zahl aus $\{0, 1, ..., k-1\}$ zuordnen und diese in einem Array speichern, welchen wir über 'Algorithmusname'.labels_ aufrufen können. Diese Zuordnung ist unter Berücksichtigungen von Permutationen natürlich nicht eindeutig. Für $k = 3$ Cluster gibt es beispielsweise $3! = 6$ verschiedene Möglichkeiten, die 3 Cluster mit jeweils mit einer eindeutigen Zahl zuzuordnen. Mit der folgenden Funktion erreichen wir deshalb, dass wir für unsere Beispieldatensätze bestehend jeweils aus 3 Gruppen jene der 6 Clusterzuordnung nehmen, die der Originallabelung am ähnlichsten ist. Dabei berechnen wir jeweils die Anzahl der Elemente,

die anders gelabelt sind als das Original und nehmen jene Zuordnung, bei der diese Anzahl am geringsten ist.

[ ]:
```
#Es gibt 6 verschiedene Möglichkeiten 3 Cluster mit den Zahlen {0,1,2} zu␣
 ↪clusterfizieren
#Wir zählen dabei immer, wie viele Elemente bei welcher Variante falsch␣
 ↪zugeordnet wurden und nehmen am Schluss
#die Variante, die der Originalzurdnung am besten entspricht

def beste_labelung (Original_label, cluster_label):

    #1. Varinate: Zuordnung wie vom Algorithmus zurückgegeben
    min = np.count_nonzero(Original_label - cluster_label)
    beste_label = cluster_label

    #2. Variante: 0->1, 1->2, 2->0:
    label_neu = -cluster_label
    label_neu[label_neu ==  0]=1
    label_neu[label_neu == -1]=2
    label_neu[label_neu == -2]=0
    if np.count_nonzero(Original_label-label_neu) < min:
        beste_label = label_neu
        min = np.count_nonzero(Original_label-label_neu)

    #3. Variante: 0->1, 1->0, 2->2:
    label_neu = -cluster_label
    label_neu[label_neu ==  0]=1
    label_neu[label_neu == -1]=0
    label_neu[label_neu == -2]=2
    if np.count_nonzero(Original_label-label_neu) < min:
        beste_label = label_neu
        min = np.count_nonzero(Original_label-label_neu)

    #4. Variante: 0->2, 1->0, 2->1:
    label_neu = -cluster_label
    label_neu[label_neu ==  0]=2
    label_neu[label_neu == -1]=0
    label_neu[label_neu == -2]=1
    if np.count_nonzero(Original_label-label_neu) < min:
        beste_label = label_neu
        min = np.count_nonzero(Original_label-label_neu)

     #5. Variante: 0->2, 1->1, 2->0:
    label_neu = -cluster_label
    label_neu[label_neu ==  0]=2
    label_neu[label_neu == -1]=1
    label_neu[label_neu == -2]=0
```

```
    if np.count_nonzero(Original_label-label_neu) < min:
        beste_label = label_neu
        min = np.count_nonzero(Original_label-label_neu)

    #6. Variante: 0->0, 1->2, 2->1:
    label_neu = -cluster_label
    label_neu[label_neu ==  0]=0
    label_neu[label_neu == -1]=2
    label_neu[label_neu == -2]=1
    if np.count_nonzero(Original_label-label_neu) < min:
        beste_label = label_neu
        min = np.count_nonzero(Original_label-label_neu)

    return(min, beste_label)
```

Nun erzeugen wir unseren ersten Datensatz, wenden k-means auf diesen an und plotten die Ergebnisse. Der Datensatz besteht dabei aus zwei Monden und einem Blob. Als Monde bezeichnen wir zwei Punktmengen, bei denen die Punkte zunächst auf zwei ineinanderliegenden Halbkreisen liegen und dann für jeden dieser Punkte in beiden Dimensionen eine normalverteilte Zufallsvariable (ZV) mit Mittelwert $\mu = 0$ und Varianz σ^2 aufaddiert wird. Im Code entspricht "noise" diesem σ. Als Blob bezeichnen wir eine Menge von zweidimensional Normalverteilten ZVn mit Erwartungswertvektor $\mu \in \mathbb{R}^2$ und Kovarianzmatrix $\Sigma = \sigma^2 I_d \in \mathbb{R}^{2x2}$. Diesen Wert für σ legen wir im Code mit "cluster_std" und den Wert für μ legen wir mit "centers" fest. Wenn wir hier k Mittelpunkte $\mu_1, \ldots, \mu_k \in \mathbb{R}^2$ festlegen, erzeugen wir k Blobs mit den jeweiligen Mittelpunkten $\mu_1, \ldots, \mu_k$ und der festlegeten Kovarianzmatrix.

[ ]:
```
#Datensatz_1: Zwei Monde und ein Blob
Monde, y_1 = datasets.make_moons(150, noise=0.1, shuffle = False,␣
 ↪random_state = 2) #erzeuge Monde
Blobs,y_2 = datasets.make_blobs(n_samples=100, centers = [[-1,-1]],␣
 ↪cluster_std = 0.25, shuffle = False, random_state = 87) #erzeuge Blob

#Füge Mode und Blob zu einem Datensatz zusammen
Daten_1 = np.concatenate((Monde, Blobs), 0)
Zugehörigkeit_1 =  np.concatenate((y_1, y_2+2), 0)

n_1 = Daten_1.shape[0]#Anzahl Datenpunkte

#Wende k-means auf den Datensatz an und plotte entsprechendes Clustering
km1 = KMeans(n_clusters=3)
km1.fit(Daten_1)
_, km_labels  = beste_labelung(Zugehörigkeit_1, km1.labels_)

#Plotte Datensatz mit korrekter Zugehörigkeit und mit k-means Clustering
fig, ax = plt.subplots(nrows=1, ncols=2,figsize=(20,8))
```

```
ax[0].set_title('Datensatz 1: Zwei Monde uns ein Blob mit korrekter␣
 ↪Zuordnung', fontsize=12, fontweight='demi')
ax[0].scatter(Daten_1[:, 0], Daten_1[:, 1], s= 50, c =Zugehörigkeit_1␣
 ↪)#Originalzugehörigkeit
ax[1].set_title('Datensatz 1: k-means Clustering', fontsize=12,␣
 ↪fontweight='demi')
ax[1].scatter(Daten_1[:, 0], Daten_1[:, 1], s= 50, c= km_labels, cmap =␣
 ↪'viridis')#k-means
fig.tight_layout()
plt.show()
```

Wir sehen, dass k-means Probleme hat, die nicht konvexen Monde zu unterscheiden. Wir werden sehen, dass wir mit der Spectralen Clusteranalyse nach geeigneter Wahl des Ähnlichkeitsgraphen und Parameter keine Probleme haben, die Cluster korrekt zu identifizieren. Dabei wollen wir auch den Einfluss der Wahl des Ähnlichkeitsgraphen und der entsprechenden Parameter beim Spectralen Clustering untersuchen: Als Algorithmus benutzen wir die Implementierung von sklearn/scikit-learn. Als Eingabeparameter geben wir die Anzahl der zu findenden Cluster ('n_clusters') und 'precomputed' beim Parameter 'affinity' an. Dies sorgt dafür, dass der Algorithmus unsere Eingabematrizen bereits als vorbereitete Ähnlichkeitsmatrizen betrachtet.

[ ]:
```
#wir Erstellen Objekt der Klasse SpectralClustering, das wir für das␣
 ↪Clustering mit kNN- und Epsilon-Graphen benutzen werden
SC_1 = SpectralClustering(n_clusters=3, affinity='precomputed',␣
 ↪assign_labels= 'kmeans')
```

Zunächst erstellen wir einen kNN-Graphen für unseren Datensatz. Dazu benutzen wir den kneighbors_graph von sklearn/scikit-learn, bei dem wir alle Punkte x_i mit Punkten x_j verbinden, wenn x_j zu den k-nächsten Nachbarn von x_i gehört. Als Rückgabewert erhalten wir eine Matrix A, die entweder eine einfache Adjazenzmatrix bestehend aus Nullen und Einsen oder eine gewichtete Adjazenzmatrix mit den Abständen $d_{ij} = \|x_i - x_j\|$ statt der Einsen ist. Dies ist noch nicht die Ähnlichkeitsmatrix, die wir beim Spectralen Clustering benutzen. Zum einen müssen wir für das Spectrale Clustering die Abstände dann noch beispielsweise mittels der Gaußschen Ähnlichkeitsfunktion zunächst in Ähnlichkeiten s_{ij} umwandeln, also

$$s_{ij} = \begin{cases} \exp\left(\frac{-{d_{ij}}^2}{2\sigma^2}\right) & \text{für } A_{ij} > 0 \\ 0 & \text{sonst} \end{cases}$$

für ein zu wählendes σ. Dies erreichen wir, indem wir einmal eine einfache ungewichtete Adjazenzmatrix des kNN-Graphen $A_{knn\_con} \in \{0,1\}^{nxn}$ und jene Matrix A_{knn_dis} mit Abständen statt der einsen berechnen. Dann wenden wir auf A_{knn_dis} eintragsweise die Gaußsche Ähnlichkeitsfunktion an. Da $\exp(0) = 1$, müssen wir die Einträge, die vorher null waren wieder auf null setzen. Dies erreichen wir durch eintragsweise Multiplikation mit A_{knn_con}. Danach müssen wir die Matrix noch symmetrisieren, da wir x_i und x_j verbinden wollen, wenn entweder x_j zu den k-nächsten Nachbarn von x_i gehört oder umgekehrt. Dies tun wir in der doppelten "for-Schleife". Auf die resultierende Matrix können wir dann Spectralen

Clusteralgorithmus anwenden. Da wir dieses Vorgehen immer wieder gebrauchen werden, schreiben wir dafür eine Funktion.

[ ]:
```
def SC_kNN_Graph (Daten, SC, k, sigma):
    n = Daten.shape[0] #Anzahl der Datenpunkte
    #Adjazenzmatrix mit Gewichten d_{ij}
    A_knn_dis = kneighbors_graph(Daten, n_neighbors=k, mode= 'distance',␣
 ↪include_self = False).toarray()

    #ungewichtete Adjazenzmatrix
    A_knn_con = kneighbors_graph(Daten, n_neighbors=k, mode=␣
 ↪'connectivity', include_self = False).toarray()

    #Wir benutzen die Gaußsche Ähnlichkeitsfkn., um Distanzen d_{ij} in␣
 ↪Ähnlichkeiten umzuwandeln
    #Wir müssen aufpassen, dass wenn A_2_knn_{ij} = 0, dann  S_2_knn_{ij}=␣
 ↪0  ebenfalls, deshalb wenden wir erst
    #Gaußkern eintragsweise an und multiplizieren dann erneut␣
 ↪eintragsweise mit A_2_knn_con, damit Einträge,
    #die null waren nun wieder null sind und nicht eins.
    S_knn = A_knn_con * np.exp(- A_knn_dis ** 2 / (2. * sigma ** 2))␣
 ↪#Ähnlichkeitsmatrix

    #Symmetrisieren der Ähnlichkeitsmatrix
    for i in range(n):
        for j in range(i):
            max = np.maximum(S_knn[i,j], S_knn[j,i])
            S_knn[i,j] = max
            S_knn[j,i] = max

    #Erstelle Graphen
    G_knn = nx.from_numpy_array(S_knn , create_using=nx.Graph)

    #Spectral Clustering mittels Ähnlichkeitsmatrix
    SC.fit(S_knn)
    labels = SC.labels_ #label

    #Gebe Graphen und label zurück
    return(G_knn, labels)
```

[ ]:
```
#Lege k für kNN fest
k_1 = 4
#Anwednung der Funktion zur Erstellung des Ähnlichkeitsgraphen und␣
 ↪Anwednung von SC
G_1_knn_1, Daten_1_knn_1_labels = SC_kNN_Graph(Daten_1, SC_1, k = k_1,␣
 ↪sigma = 0.3)
```

Als nächstes erstellen wir den ϵ- Nachbarschaftsgraphen für unseren Datensatz. Dazu be-

nutzen wir den radius_neighbors_graph von sklearn/scikit-learn, bei dem wir alle Punkte x_j mit x_i verbinden, falls $d_{ij} = \|x_i - x_j\| < \epsilon$ und als Rückgabewert die symmetrische Matrix A erhalten, für die entweder gilt

$$A_{ij} = \begin{cases} 1 & \text{für } d_{ij} < \epsilon \\ 0 & \text{sonst} \end{cases}$$

im ungewichteten Fall ('conectivity') oder

$$A_{ij} = \begin{cases} d_{ij} & \text{für } d_{ij} < \epsilon \\ 0 & \text{sonst} \end{cases}$$

im gewichteten Fall ('distance'). Wir beschränken uns auf Empfehlung von von Luxburg auf eine ungewichtete Adjazenzmatrix. Auch das folgende Vorgehen werden wir noch oft brauchen, weshalb wir hierfür eine Funktion erstellen.

[ ]:
```
def SC_eps_Graph (Daten, SC, epsilon):
    n = Daten.shape[0] #Anzahl der Datenpunkte
    ##Erstelle eps-Graphen aus Datensatz
    A_eps = radius_neighbors_graph(Daten, radius= epsilon, mode=␣
 ↪'connectivity')
    S_eps = A_eps.toarray()

    #Erstelle Graph aus der Matrix
    G_eps = nx.from_numpy_array(S_eps , create_using=nx.Graph)

    #Spectral Clustering mittels Ähnlichkeitsmatrix
    SC.fit(S_eps)
    labels = SC.labels_

    return(G_eps, labels)

#Lege Epsilon fest
epsilon_1 =0.25

#Anwednung der Funktion zur Erstellung des Ähnlichkeitsgraphen und␣
 ↪Anwednung von SC
G_1_eps_1, Daten_1_eps_1_labels = SC_eps_Graph (Daten_1, SC_1, epsilon_1)
```

[ ]:
```
#Finde für gefundene Cluster beste label und bestimme Anzahl der falsch␣
 ↪geclusterten Punkte:
anz_falsch_knn_1, Daten_1_knn_1_labels = beste_labelung(Zugehörigkeit_1,␣
 ↪Daten_1_knn_1_labels)
anz_falsch_eps_1, Daten_1_eps_1_labels = beste_labelung(Zugehörigkeit_1,␣
 ↪Daten_1_eps_1_labels)
```

```
print('Beim SC mit dem kNN-Graphen konnten wir', n_1 - anz_falsch_knn_1,␣
↪'und mit dem Epsilon-Graphen',
      n_1 - anz_falsch_eps_1, 'von', n_1, 'Datenpunkten dem Originalen␣
↪Cluster zuordnen.')

#Plotte die verschiedenen Graphen entsprechendem Clustering
#Erstelle dazu Liste mit Position jedes Knotens in der Ebene. Diese Liste␣
↪brauchen wir, um die Graphen zu plotten
pos_1= {}
for i in range (n_1):
    pos_1[i] = (Daten_1[i, 0], Daten_1[i, 1])

fig, ax = plt.subplots(nrows=2, ncols=2,figsize=(20,16))
ax[0,0].set_title('Datensatz 1: kNN Graph , $k = 4$', fontsize=12,␣
↪fontweight='demi')
nx.draw(G_1_knn_1,arrows= True, pos = pos_1,  node_size = 50,ax=ax[0,0] )
ax[0,0].set_axis_on()
ax[0,0].tick_params(left=True, bottom=True, labelleft=True,␣
↪labelbottom=True)
ax[0,0].set_xlim([-2, 2.5])
ax[0,0].set_ylim([-2, 1.5])

ax[0,1].set_title('Datensatz 1: SC mit gew. kNN Graphen, $k = 4$, $\sigma␣
↪=0.3 $', fontsize=12, fontweight='demi')
ax[0,1].scatter(Daten_1[:, 0], Daten_1[:, 1], s= 50, c=␣
↪Daten_1_knn_1_labels , cmap = 'viridis')
ax[0,1].set_axis_on()
ax[0,1].tick_params(left=True, bottom=True, labelleft=True,␣
↪labelbottom=True)
ax[0,1].set_xlim([-2, 2.5])
ax[0,1].set_ylim([-2, 1.5])

ax[1,0].set_title('Datensatz 1: Epsilon-Graph, $\epsilon = 0.2$',␣
↪fontsize=12, fontweight='demi')
nx.draw(G_1_eps_1, pos = pos_1,  node_size = 50,ax=ax[1,0] )
ax[1,0].set_axis_on()
ax[1,0].tick_params(left=True, bottom=True, labelleft=True,␣
↪labelbottom=True)
ax[1,0].set_xlim([-2, 2.5])
ax[1,0].set_ylim([-2, 1.5])

ax[1,1].set_title('Datensatz 1: SC mit Epsilon-Graphen, $\epsilon = 0.2$',␣
↪fontsize=12, fontweight='demi')
ax[1,1].scatter(Daten_1[:, 0], Daten_1[:, 1], s= 50, c=␣
↪Daten_1_eps_1_labels, cmap = 'viridis')
ax[1,1].set_axis_on()
```

```
ax[1,1].tick_params(left=True, bottom=True, labelleft=True,␣
 ↪labelbottom=True)
ax[1,1].set_xlim([-2, 2.5])
ax[1,1].set_ylim([-2, 1.5])

fig.tight_layout()
plt.show()
```

Wir erhalten für den kNN-Graphen ein sehr gutes Clusterergebnis. Für den ϵ-Graphen dagegen sehen wir, dass wir ϵ größer wählen müssen, da der Graph aus zu vielen Komponenten besteht. Das resultierende Clustering ist dementsprechend ausbaufähig. Deshalb wählen wir nun ϵ, wie von von Luxburg vorgeschlagen, als längste Kante eines minimalen Spannbaums des voll verbundenen Graphen. Dazu erstellen wir zunächst eine Funktion, die die Länge einer solchen Kante findet und diese sowie den Spannbaum und dessen Kantenmenge zurückgibt. Diese Funktion wenden wir auf unsere Daten an und wiederholen das Clustering für das neu gefundene ϵ.

[ ]:
```
def MST_Methode(Daten):
    A = euclidean_distances(Daten) #paarweise eukl. Distanzen

    #erstelle vollverbundenen Graphen aus Datensatz mit paarweisen␣
 ↪Abständen als Gewichten
    G = nx.from_numpy_matrix(A)

    #Berechne minimalen Spannbaum mittels Kruskal Algorithmus
    min_spann_baum = nx.tree.minimum_spanning_edges(G,␣
 ↪algorithm="kruskal", data=False)
    kantenliste = list(min_spann_baum) #Kanntenmenge des Spannbaums


    #längste Kante in minimalem Spannbaum
    max = 0
    for kante in kantenliste:
        if A[kante] >= max:
            max = A[kante]

    return(max, kantenliste, G)

#epsilon durch MST-Methode
epsilon_2, kantenmenge, G = MST_Methode(Daten_1)
print('Das kleinste Epsilon, für das wir einen zusammenhängenden Graphen␣
 ↪erhalten ist: ', epsilon_2)

#Plotte Spannbaum
fig, ax = plt.subplots(figsize=(9,7))
ax.set_title('MST von Datensatz 1', fontsize=12, fontweight='demi')
nx.draw(G, pos = pos_1, edgelist =kantenmenge, node_size = 40)
```

```
ax.set_axis_on()
ax.tick_params(left=True, bottom=True, labelleft=True, labelbottom=True)
ax.set_xlim([-2, 2.5])
ax.set_ylim([-2, 1.5])

fig.tight_layout()
plt.show()
```

Ab jetzt verzichten wir auf die Code zur Erstellung der Plots. Die Erstellung dieser erfolgt analog zu oben.

Für den ϵ-Graphen mit ϵ aus der MST-Methode erhalten wir ein gutes Clustering.

Als nächstes betrachten wir als Ähnlichkeitsgraphen den voll-verbundenen Graphen, bei dem wir alle Knoten durch Kanten miteinander verbinden und diese durch Ähnlichkeitswerte s_{ij} gewichten. Die Gewichte erhalten wir durch die Gaußsche Ähnlichkeitsfunktion, also

$$s_{ij} = \exp\left(\frac{-{d_{ij}}^2}{2\sigma^2}\right) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right),$$

wobei wir σ versuchen werden, geeignet zu wählen. Dies erreichen wir, indem wir für den Parameter 'affinity' den Wert 'rbf' wählen, wobei 'rbf' für 'radial basis function' kernel steht. Diese Funktion ist implementiert als

$$\exp\left(-\gamma\|x_i - x_j\|^2\right).$$

Dies entspricht mit $\gamma = \frac{1}{2\sigma^2}$ also genau der Definition der Gaußschen Ähnlichkeitsfunktion. Dem Spectralen Clustering Algorithmus übergeben wir einem Wert für γ, wobei $\gamma \geq 1$ sein muss. Also müssen wir $\sigma \leq \sqrt{\frac{1}{2}}$ wählen.

Als erstes wählen wir $\sigma = \epsilon$ für das ϵ aus der MST-Methode von oben.

[ ]:
```
sigma_1 = epsilon_2
gamma_1 = 1/ (2* (sigma_1)**2)

#SC mit vollverbundenem Graphen
SC_2 = SpectralClustering(n_clusters=3,  gamma=gamma_1, affinity='rbf', ␣
↪assign_labels='kmeans')
SC_2.fit(Daten_1)
Daten_1_voll_1_labels= SC_2.labels_
anz_falsch_voll_1, Daten_1_voll_1_labels = beste_labelung(Zugehörigkeit_1,␣
↪Daten_1_voll_1_labels)
```

Wir sehen, dass für $\sigma = \epsilon$ für den Datensatz nicht optimal ist. Der Spectrale Cluster Algorithmus kann die beiden Monde nicht richtig unterscheiden. Die lässt vermuten, dass Knoten, die "weiter" voneinander entfernt sind, noch ein zu hohes Gewicht haben, also σ zu groß gewählt ist. Wir versuchen deshalb im zweiten Versuch, σ durch die Mittelwertmethode zu wählen.

```
[ ]: def Mittelwertmethode (Daten, n):
         #Berechne kNN-Matrix mit Distanzen mit k wie oben angegeben
         k = int(np.ceil(np.log(n) +1)) #aufgerundetes k damit k integer

         #Adjazenzmatrix für knn-Graphen
         A_1_knn_dis = kneighbors_graph(Daten, n_neighbors=k, mode= 'distance').
     ↪toarray()

         #Mittelwert aller Einträge != 0 für jede Zeile, dies Eentspricht dem␣
     ↪mittleren Abstand pro Knoten
         #zu seinen k nächsten Nachbarn
         zeilen_mittelwert = np.sum(A_1_knn_dis, axis=1) /k

         #mittelwert dieser Abstände
         mittelwert = np.mean(zeilen_mittelwert)

         return(mittelwert)


     #Anwendung auf Daten:
     sigma_2 = Mittelwertmethode(Daten_1, n_1)
     print('Sigma durch die Mittelwertmethode:', sigma_2) #sigma = 0.138

     gamma_2 = 1/ (2* (sigma_2)**2)
     SC_2 = SpectralClustering(n_clusters=3,  gamma=gamma_2, affinity='rbf', ␣
     ↪assign_labels='kmeans')
     SC_2.fit(Daten_1)
     Daten_1_voll_2_labels= SC_2.labels_
     anz_falsch_voll_2, Daten_1_voll_2_labels = beste_labelung(Zugehörigkeit_1,␣
     ↪Daten_1_voll_2_labels)
```

Mit diesem σ sehen wir, dass das entsprechende Clustering die Monde und den Blob sehr gut voneinander unterscheidet. Wir haben es geschafft, alle Punkte richtig zu clustern. Dies motiviert das gleiche σ und k auch für den kNN-Graphen zu benutzen.

```
[ ]: #Erstelle kNN-Graphen für k und sigma aus Mittelwertmethode
     k_2 =  int(np.ceil(np.log(n_1) +1)) #k = 7
     G_1_knn_2, Daten_1_knn_2_labels = SC_kNN_Graph(Daten_1, SC_1, k = k_2,␣
     ↪sigma = sigma_2)

     #nehme jene Label, die der tatsächlichen labelung am ähnlichsten ist
     anz_falsch_knn_2, Daten_1_knn_2_labels = beste_labelung(Zugehörigkeit_1,␣
     ↪Daten_1_knn_2_labels)
```

Wie wir sehen, konnten wir auch hier alle Punkte richtig in die Cluster einteilen, aus denen sie eigentlich stammen. Der kNN-Graph hat gegenüber dem voll-verbundenem Graphen den Vorteil, dass die resultierende Ähnlichkeitsmatrix dünn besetzt ist, was gerade für große Datenmengen für die numerische Berechnung der Eigenvektoren von großem Vorteil ist.

Nun untersuchen wir die verschiedenen Ähnlichkeitsgraphen anhand eines anderen Datensatzes. Dazu erzeugen wir zunächst 500 Punkte aufgeteilt in drei Blobs mit $\mu_1 = \begin{pmatrix} -8.95 \\ -5.46 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -4.59 \\ 0.09 \end{pmatrix}$ und $\mu_3 = \begin{pmatrix} 1.94 \\ 0.51 \end{pmatrix}$ und $\Sigma_1 = \Sigma_2 = \Sigma_3 = I_d$.

Also konkret: $Y_1, \ldots, Y_{167} \sim \mathcal{N}_2((-8.95\ -5.46)^\intercal, I_d)$, $Y_{168}, \ldots, Y_{334} \sim \mathcal{N}_2((-4.59\ 0.09)^\intercal, I_d)$, $Y_{335}, \ldots, Y_{500} \sim \mathcal{N}_2((1.94\ 0.51)^\intercal, I_d)$. Wir tranformieren die Punkte dann durch Multiplikation mit

$$A = \begin{pmatrix} 0.5 & -0.4 \\ -0.5 & 0.7 \end{pmatrix} \tag{2}$$

und erhalten $X_1, \ldots, X_{500}$ mit $X_i = AY_i$.

[ ]:
```
n_2 = 500 #Anzahl Datenpunkte

#Blobs
Daten_2, Zugehörigkeit_2 = make_blobs(n_samples=n_2,  random_state=170,
                                      centers= [[-8.94, -5.46],␣
↪[-4.59, 0.09], [ 1.94,  0.51]], shuffle= False)
#transponierte Transformationsmatrix
transformation = [[0.5, -0.5], [-0.4, 0.7]]

#transformierte Daten
Daten_2 = np.dot(Daten_2, transformation) #entspricht Y^T * A^T, da␣
↪Datenpunkte Zeilenvektoren sind

#Anwendung kmeans
km = KMeans(n_clusters=3)
km.fit(Daten_2)
_ , km_labels = beste_labelung(Zugehörigkeit_2, km.labels_)
```

Als nächstes wollen wir auch hier den Einfluss der Wahl des Ähnlichkeitsgraphen und der entsprechenden Parameter beim Spectralen Clustering untersuchen. Wir wandeln unsere Daten dazu wieder in verschiedene Ähnlichkeitsgraphen um. Wir berechnen zunächst die gewichtete Ähnlichkeitsmatrix des kNN-Graphen für $k = \lceil \log(n) + 1 \rceil = 8$ und $\sigma \approx 0.151$ aus der Mittelwertmethode und wenden Spectrales Clusetring an. Dabei gehen wir analog wie beim ersten Datensatz vor.

[ ]:
```
#Werte für k und sigma
k_1 =  int(np.ceil(np.log(n_2) +1))
sigma_1 = Mittelwertmethode(Daten_2, n_2)
print('k:', k_1,', sigma:', sigma_1)

G_2_knn_1, Daten_2_knn_1_labels = SC_kNN_Graph (Daten_2, SC_1, k_1, sigma␣
↪= sigma_1)
```

Da wir für den ersten Datensatz mit ϵ aus der MST-Methode gute Ergebnisse erhalten haben, wählen wir zunächst auch hier wieder ϵ durch die MST-Methode.

```
[ ]: epsilon_1, kantenmenge, G = MST_Methode(Daten_2)
print('Das kleinste Epsilon, für das wir einen verbundenen Graphen␣
 ↪erhalten, ist: ', epsilon_1)

epsilon_1 = round(epsilon_1, 4)  #Wir runden epsilon auf 4␣
 ↪Nachkommastellen
G_2_eps_1, Daten_2_eps_1_labels = SC_eps_Graph (Daten_2, SC_1, epsilon_1)
```

```
[ ]: #Zum Plotten der verschiedenen Graphen benötigen wir auch hier Liste mit␣
 ↪Postition in der Ebene für jeden Knoten
pos_2= {}
for i in range (n_2):
    pos_2[i] = (Daten_2[i, 0], Daten_2[i, 1])

#beste Label für knn- und eps-Graphen
anz_falsch_knn_1, Daten_2_knn_1_labels = beste_labelung(Zugehörigkeit_2,␣
 ↪Daten_2_knn_1_labels)
anz_falsch_eps_1, Daten_2_eps_1_labels = beste_labelung(Zugehörigkeit_2,␣
 ↪Daten_2_eps_1_labels)
```

Wir sehen, dass wir mit dem kNN-Graphen für $k = \lceil \log(n) + 1 \rceil = 8$ zunächst einen zusammenhängenden Graphen zusammen mit einem sehr gutem Clustering erhalten. Bis auf einen Punkt konnten alle Punkte richtig zugeordnet werden. Für den ϵ-Graphen mit ϵ aus der MST-Methode ist zu beobachten, dass selbst der Ausreißerpunkt oben links mitverbunden ist. Allerdings sehen wir auch, dass dadurch die oberen beiden Blobs nun sehr stark miteinander verbunden sind. Dies führt dazu, dass das resultierende Clustering diese Punktmengen nicht komplett unterscheiden kann.

Wir wollen nun herausfinden, wie sie das Clustering für die jeweiligen Graphen ändert, wenn wir die Parameter anpassen. Für den ϵ-Graphen wollen wir nun ein kleiners ϵ wählen. Wir versuchen $\epsilon = 0.5$.

```
[ ]: ##Erstelle eps-Graphen aus Datensatz 2 mit kleinerem Epsilon
epsilon_2 = 0.5

#Erstelle Graph aus der Matrix
G_2_eps_2, Daten_2_eps_2_labels = SC_eps_Graph (Daten_2, SC_1, epsilon_2)
```

Für den ϵ-Graphen mit $\epsilon = 0.5$ sehen wir, dass das Graph nicht zusammenhängend ist. Der Ausreißerknoten oben links ist nicht verbunden. Da die restlichen Knoten des Graphen ansonsten zusammenhängend sind, ist das resultierende Clustering dennoch gut. Lediglich der nicht verbundene Knoten und zwei andere Knoten werden falsch zugeordnet.

Von Luxburg empfiehlt zwar den ϵ-Graphen ungewichtet zu betrachten, wir erstellen an dieser Stelle aber dennoch eine Funktion, mit der man eine gewichtete Version des ϵ-Graphen erhält. Die Gewichtung erfolgt ebenfalls mit der Gaußschen Ähnlichkeitsfunktion mit Parameter σ. Diese Funktion angewandt auf ϵ aus der MST-Methode und σ durch die Mittelwertmethode führt für unseren Beispieldatensatz 2 aufgrund der Gewichtung zu sehr guten Clusterergebnissen. Deshalb empfehlen wir, falls man den $\epsilon - Graphen$ benutzen möchte,

eine gewichtete Version dieses Graphen in Betracht zu ziehen.

[ ]:
```
#eps-Ähnlichkeitsgraph gewichtet
def SC_eps_Graph_gew (Daten, SC, epsilon, sigma):
    #Anzahl der Datenpunkte
    n = Daten.shape[0]

    A_knn_dis = radius_neighbors_graph(Daten, radius= epsilon,  mode=␣
 ↪'distance').toarray()  #Adjazenzmatrix mit Gewichten d_{ij}
    A_knn_con = radius_neighbors_graph(Daten, radius= epsilon,  mode=␣
 ↪'connectivity').toarray() #ungewichtete Adjazenzmatrix

    #Wir benutzen die Gaußsche Ähnlichkeitsfkn., um Distanzen d_{ij} in␣
 ↪Ähnlichkeiten umzuwandeln
    #Wir müssen aufpassen, dass wenn A_2_knn_{ij} = 0, dann  S_2_knn_{ij}=␣
 ↪0  ebenfalls, deshalb wenden wir erst
    #Gaußkern eintragsweise an und multiplizieren dann erneut␣
 ↪eintragsweise mit A_2_knn_con, damit Einträge, die null waren
    #nun wieder null sind und nicht eins.
    S_knn = A_knn_con * np.exp(- A_knn_dis ** 2 / (2. * sigma ** 2))␣
 ↪#Ähnlichkeitsmatrix

    #Symmetrisieren der Ähnlichkeitsmatrix
    for i in range(n):
        for j in range(i):
            max = np.maximum(S_knn[i,j], S_knn[j,i])
            S_knn[i,j] = max
            S_knn[j,i] = max

    #Erstelle Graphen
    G_knn = nx.from_numpy_array(S_knn , create_using=nx.Graph)

    #Spectral Clustering mittels Ähnlichkeitsmatrix
    SC.fit(S_knn)

    #label
    labels = SC.labels_

    #Gebe Graphen und label zurücl
    return(G_knn, labels)

G_2_eps_2, Daten_2_eps_2_labels = SC_eps_Graph_gew (Daten_2, SC_1,␣
 ↪epsilon_1, sigma = sigma_1)

anz_falsch_eps_2, Daten_2_eps_2_labels = beste_labelung(Zugehörigkeit_2,␣
 ↪Daten_2_eps_2_labels)
```

Nun überprüfen wir, ob wir k noch kleiner wählen können, ohne dass der Graph in mehrere

Zusammenhangskomponenten zerfällt. Dies hat den Vorteil, dass die resultierende Ähnlichkeitsmatrix dünner besetzt ist. Dazu probieren wir verschiedene Werte für k aus. Wir starten mit $k = 4$ und werden sehen, dass dies auch der kleinste Wert ist, den wir wählen können. Für $k = 3$, zerfällt der Graph. Den Wert für σ erhöhen wir im Vergleich zum kNN Graphen von oben auf 0.2, da wir weniger Knoten miteinander verbinden.

[ ]:
```
##Wir berechnen zunächst die die gewichtete Adjazenzmatrix des kNN-Graphen
k_2 =  4
k_3 = 3
G_2_knn_2, Daten_2_knn_2_labels = SC_kNN_Graph (Daten_2, SC_1, k_2, sigma␣
↪= 0.2)
G_2_knn_3, Daten_2_knn_3_labels = SC_kNN_Graph (Daten_2, SC_1, k_3, sigma␣
↪= 0.2)
```

Für $k = 4$ erhalten wir sowohl noch einen zusammenhängenden Graphen als auch ein gutes Clustering. Für $k = 3$ dagegen sieht das anders aus. Der Graph ist nun nicht mehr zusammenhängend. Ein teil des obersten Blobs ist abgespalten vom Rest und wird durch den Algorithmus auch als eigenes Cluster identifiziert. Wir müssen also $k > 3$ für diesen Datensatz wählen.

Als nächstes betrachten wir als Ähnlichkeitsgraphen den voll-verbundenen Graphen, bei dem wir alle Knoten durch Kanten miteinander verbinden und diese durch Ähnlichkeitswerte s_{ij} gewichten. Die Gewichte erhalten wir wieder durch die Gaußsche Ähnlichkeitsfunktion, also

$$s_{ij} = \exp\left(\frac{-{d_{ij}}^2}{2\sigma^2}\right) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right).$$

Ein erster Ansatz ist es σ wie beim ersten Datensatz durch $\sigma = \epsilon$ für das ϵ aus der MST-Methode zu wählen. Dies funktioniert hier leider nicht, da wir $\sigma \leq \sqrt{\frac{1}{2}}$ wählen müssen und $\epsilon = 0.906$. Deshalb wählen wir für σ zunächst den größtmöglichen Wert, den die Implementierung von Sklearn zulässt, nämlich $\sigma = \sqrt{\frac{1}{2}} \approx 0.707$.

[ ]:
```
sigma_1 = 1/(np.sqrt(2))
gamma_1 = 1/ (2* (sigma_1)**2)

SC_2 = SpectralClustering(n_clusters=3,  gamma=gamma_1, affinity='rbf', ␣
↪assign_labels='kmeans')
SC_2.fit(Daten_2)
Daten_2_voll_1_labels = SC_2.labels_
anz_falsch_voll_1, Daten_2_voll_1_labels = beste_labelung(Zugehörigkeit_2,␣
↪Daten_2_voll_1_labels)
```

Wie wir sehen, müssen wir σ kleiner wählen. Der Algorithmus kann die beiden oberen Blobs nicht auseinander halten. Das Clustering ist ähnlich schlecht , wie durch k-means. Wir verwenden deshalb wie beim ersten Datensatz wieder die Mittelwertmethode an.

[ ]:
```
sigma_2 = Mittelwertmethode(Daten_2, n_2)
print('Sigma durch die Mittelwertmethode:', sigma_2) #= 0.1512
```

```
gamma_2 = 1/ (2* (sigma_2)**2)

SC_2 = SpectralClustering(n_clusters=3,  gamma=gamma_2, affinity='rbf', ␣
 ↪assign_labels='kmeans')
SC_2.fit(Daten_2)
Daten_2_voll_2_labels= SC_2.labels_
anz_falsch_voll_2, Daten_2_voll_2_labels = beste_labelung(Zugehörigkeit_2,␣
 ↪Daten_2_voll_2_labels)
```

Wir sehen, dass wir auch hier wie beim ersten Datensatz ein sehr gutes Ergebnis erhalten.

Im Allgemeinen hat sich bei den beiden Datensätzen gezeigt, dass der kNN-Graph und der voll-verbundene Graph zusammen mit der Mittelwertmethode die stabilsten Ergebnisse lieferten. Der kNN-Graph hat gegenüber dem voll-verbundenem Graphen allerdings noch den Vorteil, dass die resultierende Ähnlichkeitsmatrix dünn besetzt ist. Nun wollen wir überprüfen, wie sich der kNN-Graph mit Parametern der Mittelwertmethode bei einem "echten" Datensatz schlägt.

Anwendungen auf Iris-Datensatz: Abb. 8
Wir wollen nun Spectrales Clustering mittel des kNN-Graphen auf den Iris-Datensatz von Sir Ronald Fischer (1936) anwenden. Dieser vierdimensionale Datensatz besteht aus jeweils 150 Datenpunkten, wobei jeweils 50 Datenpunkte einer von drei verschiedenen Iris-Spezies (Iris setosa, Iris virginica und Iris versicolor) angehören. Jeder Datenpunkt hat vier Attribute: Sepal length, Sepal width, Petal length und Petal width. Wir beginnen wir dem Import des Datensatzes.

[ ]:
```
#Wir laden den Datensatz
iris = datasets.load_iris()
Daten = iris.data  #Daten aus dem R^4
Zugehörigkeit = iris.target #Originale Zugehörigkeit zu den 3 Arten
```

[ ]:
```
#Wähle k und sigma durch Mittelwertmethode
k_1 =  int(np.ceil(np.log(Daten.shape[0]) +1))
sigma_1 = Mittelwertmethode(Daten, Daten.shape[0])

print('k = ', k_1, ', sigma = ', sigma_1)

G_knn_1, Daten_knn_labels_1 = SC_kNN_Graph(Daten, SC_1, k = k_1, sigma =␣
 ↪sigma_1)

Anzahl_falsch, Daten_knn_labels_1 = beste_labelung(Zugehörigkeit,␣
 ↪Daten_knn_labels_1)
```

[ ]:
```
#k-means Clustering zum Vergleich:
km = KMeans(n_clusters=3)
km.fit(Daten)

Anzahl_falsch_km , km_labels = beste_labelung(Zugehörigkeit, km.labels_)
```

```
print(150 -Anzahl_falsch, 'der 150 Punkte wurden durch das Spectrale␣
↪Clustering den 3 Sorten richtig zugeordnet.\n',
      Anzahl_falsch, ' Punkte wurden falsch zugeordnet, dies entspricht',␣
↪Anzahl_falsch/150 * 100, '% aller Punkte.\n'
     ' Zum Vergleich: Durch k-means wurden ' , 150 - Anzahl_falsch_km, '␣
↪Datenpunkte richtig und' , Anzahl_falsch_km,
     ' Datenpunkte falsch zugeordnet.')
```

Nun wollen wir noch genauer identifizieren, welche Punkte falsch zugeordnet wurden. Dazu zählen wir jeweils, wie viele Pflanzen jeder Sorte welchem Cluster zugeordnet wurden. Betrachten wir dazu kurz die Originalzugehörigkeit der Daten:

[ ]:
```
print(Zugehörigkeit)
print(iris.target_names)
```

Wir sehen, dass die ersten 50 Punkte der Spezies “iris setosa”, die nächsten 50 der Spezies “iris versicolor” und die letzten 50 der Spezies “iris virginica” angehören. Wir zählen nun in drei Arrays, bestehend jeweils aus drei Einträgen, wie viele Datenpunkte der jeweiligen Spezies, welchem Cluster zugeordnet wurden und erzeugen aus diesen eine Tabelle.

[ ]:
```
setosa_cluster = [0,0,0]
versicolor_cluster = [0,0,0]
virginica_cluster = [0,0,0]

setosa_labels = Daten_knn_labels_1[:50] #ersten 50 Einträge von␣
↪label-Array entsprechen Labelung für die setosa Spezies
versicolor_labels = Daten_knn_labels_1[50:100] #Einträge 51 bis 100␣
↪entsprechen Labelung für die versicolor Spezies
virginica_labels = Daten_knn_labels_1[100:150] #Einträge 101 bis 150␣
↪entsprechen Labelung für die virginica Spezies

setosa_cluster[0] = len(setosa_labels[setosa_labels == 0]) #setosa␣
↪Pflanzen, die Cluster 0 zugeordnet wurden
setosa_cluster[1] = len(setosa_labels[setosa_labels == 1]) #setosa␣
↪Pflanzen, die Cluster 1 zugeordnet wurden
setosa_cluster[2] = len(setosa_labels[setosa_labels == 2]) #setosa␣
↪Pflanzen, die Cluster 2 zugeordnet wurden

versicolor_cluster[0] = len(versicolor_labels[versicolor_labels == 0])␣
↪#versicolor Pflanzen, die Cluster 0 zugeordnet wurden
versicolor_cluster[1] = len(versicolor_labels[versicolor_labels == 1])␣
↪#versicolor Pflanzen, die Cluster 1 zugeordnet wurden
versicolor_cluster[2] = len(versicolor_labels[versicolor_labels == 2])␣
↪#versicolor Pflanzen, die Cluster 2 zugeordnet wurden
```

```
virginica_cluster[0] = len(virginica_labels[virginica_labels == 0])␣
 ↪#virginica Pflanzen, die Cluster 0 zugeordnet wurden
virginica_cluster[1] = len(virginica_labels[virginica_labels == 1])␣
 ↪#virginica Pflanzen, die Cluster 1 zugeordnet wurden
virginica_cluster[2] = len(virginica_labels[virginica_labels == 2])␣
 ↪#virginica Pflanzen, die Cluster 2 zugeordnet wurden

#Erstelle Ergebnisstabelle
Spalten = ( 'Cluster 0', 'Cluster 1', 'Cluster 2')
Zeilen = ('setosa' , 'versicolor', 'virginica')
cellText = [setosa_cluster, versicolor_cluster, virginica_cluster]

fig, ax = plt.subplots()
table = ax.table(cellText=cellText, loc='center', rowLabels=Zeilen,␣
 ↪colLabels=Spalten)
table.set_fontsize(12)
table.scale(1,4)
ax.axis('off')
fig.tight_layout()
plt.show()
```

Wir sehen, dass wir mittels Spectralem Clustering die 'setosa' Spezies korrekt einem Cluster zuordnen konnten. Die 'versicolor' und 'virginica' Spezies konnten wir dagegen nicht perfekt trennen. Die 'versicolor'-Pflanzen befinden sich zwar alle im gleichen Cluster, gleichzeitig befinden sich in diesem aber auch 14 Pflanzen der Gattung 'virginica'.

Vergleich von k-means und Spectralem Clustering bei nicht konvexen Daten
Zum Abschluss erzeugen wir noch das Einstiegsbeispiel aus Abbildung 1, an dem wir in der Arbeit die Problematik von k-means, nur konvexe Cluster zu finden, erläutern. Dazu erzeugen wir einen Datensatz der aus einem großen und zwei kleinen Kreisen besteht.

Nun erzeugen wir den Datensatz bestehend aus 500 Punkten $X_1, \dots, X_{500}$ und plotten diesen anschließend. Wir benutzen dazu die Funktion 'make_cicles' von sklearn. Diese erzeugt zunächst $Y_1, \dots, Y_{250}$ und $Z_1, \dots, Z_{250}$ mit

$$Y_i = \begin{pmatrix} \cos((i-1)\frac{2\pi}{249}) \\ \sin((i-1)\frac{2\pi}{249}) \end{pmatrix} \tag{3}$$

und

$$Z_i = 0.8 \begin{pmatrix} \cos((i-1)\frac{2\pi}{249}) \\ \sin((i-1)\frac{2\pi}{249}) \end{pmatrix}, \tag{4}$$

also jeweils 250 Punkte, die auf einem Kreis mit Radius 1 und 250 Punkte, die auf einem Kreis mit Radius 0.8 liegen. Danach definiert die Funktion $X_1, \dots X_{250} = Y_1, \dots, Y_{250}$ und $X_{251}, \dots X_{500} = Z_1, \dots, Z_{250}$ und addiert für jedes X_i in beiden Dimensionen eine normalverteilte Zufallsvariable mit Mittelwert $\mu = 0$ und Varianz σ^2 auf. Den Wert für σ legen wir dabei mit dem Parameter 'noise' fest.

```
[ ]: random_state = 1 #Wir wählen einen random_state, damit Ergebnisse␣
     ↪reproduzierbar sind
     n = 500 #Anahl der Datenpunkte

     Kreise, zugekörigkeit = datasets.make_circles(n_samples=n, factor=0.5,␣
     ↪noise=0.07, shuffle= False,
                                                   random_state= random_state)
```

Ziel beim k-means-Clustering ist es, eine Partition S_1, S_2 mit Zentren μ_1, μ_2 zu finden, die

$$\sum_{x_i \in S_1} \|x_i - \mu_l\|^2 + \sum_{x_i \in S_2} \|x_i - \mu_l\|^2 \tag{5}$$

minimiert. Dazu wählen wir in jedem Schritt Zentren μ_1, μ_2, ordnen jedem Punkt, dem Cluster zu, dessen Mittelpunkt am nächsten ist und aktualisieren die Mittelpunkte als Durchschnitt der Datenpunkte pro Cluster. Bedingt durch die Minimierung der Abstände zu den Zentren findet k-means dabei nur konvexe Cluster. Für diesen Datensatz wird deutlich, dass k-means die Kreise jeweils, wie einen Kuchen in der Mitte durchschneidet. Für dieses Beispiel ist er also ungeeignet.

```
[ ]: kmeans = KMeans(n_clusters=2).fit_predict(Kreise) #Label für jeden␣
     ↪Datenpunkt
```

Wir werden nun sehen, dass wir die Daten mittels Spectralem Clustering (SC) korrekt zuordnen können. Wir benutzen für das SC den kNN-Graphen mit Parametern k und σ, die wir durch die Mittelwertmethode wählen.

```
[ ]: k = int(np.ceil(np.log(n) +1))
     sigma = Mittelwertmethode(Kreise, n)
     print('k = ', k , '\n', 'sigma = ', sigma)

     #Wir verwenden hier als Ähnlichkeitsgraphen den kNN-Graphen mit k und␣
     ↪sigma durch die Mittelwertmethode
     SC = SpectralClustering(n_clusters=2, affinity='precomputed' )
     G_knn, SC_labels = SC_kNN_Graph(Kreise, SC, k = k, sigma = sigma)
```

Wir haben bereits gesehen, dass beim Clustering werden Datenpunkte in k Cluster gruppiert, indem jedem Punkt eine Zahl (label) $\in \{0, 1, \ldots, k-1\}$ zugeordnet wird, die angibt in welchem Cluster er sich befindet. Im Folgenden berechnen wir die Anzahl der Punkte, die falsch zugeordnet wurden. Da wir nicht wissen, welches Cluster der Algorithmus mit welcher Zahl identifiziert, betrachten wir auch in den invertierten Fall und nehmen jene Labelung, die mit der Originallabelung am besten übereinstimmt (analog zum fall von drei Clustern (s. Funktion beste_Labelung).

```
[ ]: SC_labels_inv = abs(SC_labels -1 ) #invertierte labels

     #Falsch zugeordneten Punkte
     anz_falsch = min( sum( abs(zugekörigkeit- SC_labels)), sum(␣
     ↪abs(zugekörigkeit- SC_labels_inv)))
```

```
print('Wir konnten ', n- anz_falsch, 'von ', n, 'Punkten richtig clustern.
 ↪')
```

Mit Spectralem Clustering haben wir alle Punkte richtig zugeordnet.